Pronalazak rješenja platformske igre primjenom strojnog učenja

Katić, Ana-Marija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

PRONALAZAK RJEŠENJA PLATFORMSKE IGRE PRIMJENOM STROJNOG UČENJA

Diplomski rad

Ana-Marija Katić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 29.08.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Ana-Marija Katić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1209R, 07.10.2021.
OIB studenta:	65275575583
Mentor:	izv. prof. dr. sc. Časlav Livada
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 1:	izv. prof. dr. sc. Časlav Livada
Član Povjerenstva 2:	dr. sc. Marija Habijan
Naslov diplomskog rada:	Pronalazak rješenja platformske igre primjenom strojnog učenja
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu je potrebno napraviti funkcionalnu platformsku igru s elementima izmjene gravitacije. Zatim je potrebno koristeći ML-agents alate napraviti model za pronalazak rješenja razina različitih težina. Tema rezervirana za: Ana-Marija Katić
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	29.08.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 08.09.2023.

Ime i prezime studenta:	Ana-Marija Katić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1209R, 07.10.2021.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Pronalazak rješenja platformske igre primjenom strojnog učenja**

izrađen pod vodstvom mentora izv. prof. dr. sc. Časlav Livada

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Umjetna inteligencija relativno je nova disciplina čiji razvoj počinje krajem Drugog Svjetskog Rata. Prvim teorijskim radom stvaranja umjetne inteligencije uzima se članak „Computing Machinery and Intelligence“ Alana Mathisona Turinga, objavljen 1950. godine. Turing je u tom članku opisao test prema kojem se ispituje inteligencija nekog stroja, poznat kao Turingov Test, te je uveo pojmove strojno učenje, genetski algoritmi i podržano učenje [1].

Umjetna inteligencija [2] predstavlja sposobnost oponašanja ljudskih aktivnosti poput donošenja odluka, učenja, planiranja i predviđanja. Ona kao dio računarstva ima široku i raznovrsnu primjenu u raznim područjima od robotike, računalnih igara i simulacija, generiranja teksta, stvaranja glazbe i slika, računalnog vida, prevođenja, ekspertnih sustava i drugo. Osim primjena u računarstvu ona ima utjecaj na sve aspekte života i gospodarstva među kojima se koristi za prepoznavanje infekcija uz pomoć višeslojne kompjuterizirane tomografije toraksa u medicini, za autonomnu vožnju u automobilskom i željezničkom prometu te izgradnju održivog prehrambenog sustava.

Iako strojno učenje [3] predstavlja samo dio primjene umjetne inteligencije ono ima veliki utjecaj na kojemu se temelji njen razvoj. „Algoritmi strojnog učenja „uče“ informacije i odnose među njima izravno iz podataka ne oslanjajući se na teorijske jednadžbe i matematičke modele. “ [4] Ti algoritmi postaju sve bolji povećanjem korištene količine podataka, odnosno brojem uzoraka koje mogu koristiti za učenje. Stoga je primjena strojnog učenja sve raznovrsnija.

U diplomskom radu opisan je razvoj i pronalazak rješenja platformske igre primjenom strojnog učenja. Za izradu platformske igre korišten je Unity višeplatformski (engl. *cross-platform*) softver s alatom za strojno učenje Unity Machine Learning Agents Toolkit (u daljnjem tekstu ML-Agents). U svrhu treniranja agenta za pronalazak rješenja razina korišteno je podržano učenje. Konačan cilj je stvoriti agenta koji može reagirati na promjene u svojoj okolini i ispravno postupiti na temelju njih.

Rad se nastavlja s pregledom područja teme rada, odnosno opisom aktualnih dosega u području rada. Zatim su opisane korištene tehnologije i objašnjeni elementi strojnog učenja za njegovu izradu. Nakon toga opisana je izrada projekta, rezultati treniranja agenta, te su dani krajnji komentari na rezultate platformske igre.

1.1. Zadatak diplomskog rada

U radu je potrebno napraviti funkcionalnu platformsku igru s elementima izmjene gravitacije. Zatim je potrebno koristeći ML-Agents alate napraviti model za pronalazak rješenja razina različitih težina.

2. PREGLED PODRUČJA RADA

Razvoj igara već se desetak godina nalazi u prvom planu razvoja umjetne inteligencije, ali se u zadnjih par godina strojno učenje počinje koristiti s ciljem razvoja igara s responzivnim, zanimljivim i dojmljivim iskustvom za korisnike [5]. Neke od prednosti koje strojno učenje pruža u razvoju video igara programerima su: stvaranje realističnijih i impresivnijih elemenata koji poboljšavaju korisničko iskustvo, otkrivanje zlouporabe i sprječavanje hakera, te poboljšanje stvaranja sadržaja igre. S druge strane postoje prednosti koje poboljšavaju korisničko iskustvo, a to su sve realističniji likovi u igri kojima upravlja računalo (engl. *non-playable character*), bolja interakcija s drugim igračima, estetski ugodniji vizualni prikazi, niža predvidivost razina igre i drugo [6].

Dobar primjer korištenja strojnog učenja i umjetne inteligencije bila bi igra No Man's Sky [5] koja koristi proceduralno stvaranje sadržaja (engl. *procedural content generation*) za kreiranje unikatnog iskustva pojedinačnim korisnicima. Korištenje proceduralnog stvaranja sadržaja omogućuje kreiranje beskonačno mnogo jedinstvenih svemira s velikim brojem planeta koje korisnici mogu istraživati.

Osim proceduralnog stvaranja sadržaja strojnim učenjem može se poboljšati ponašanje likova u igri kojima upravlja računalo. Primjer toga bi bili neprijatelji u igri Shadow of Mordor [5]. U ovoj igri neprijatelji pamte prijašnja ponašanja, odnosno ishode bitki. Ovisno o ishodu bitke rang neprijatelja se mijenja. To se postiže praćenjem podataka o susretima i na temelju tih podataka proceduralno se stvaraju neprijatelji, a to stvara jedinstveno i zanimljivo iskustvo korisnicima.

Slično prethodnoj igri Shadow of Mordor programeri igre Red Dead Redemption 2 [5] htjeli su postići realistično ponašanje likova kojima upravlja računalo. To su postigli stvaranjem svih likova koji imaju svoje živote. Njihovi životi ne ovise o igraču, odnosno oni ne čekaju na jednom mjestu kako bi mogli komunicirati s igračem. Ponašanje likova kojima upravlja računalo ovisi o njihovim promatranjima (engl. *observations*) vezanima za igrača i na temelju njih biraju svoje akcije.

Uz druge programe za rad s umjetnom inteligencijom i strojnim učenjem poput OpenAI koristi se i Unity s alatom ML-Agents. ML-Agents omogućuje kreiranje okoline u kojoj se podučavaju inteligentni agenti da sami uče kombinacijom dubokog podržanog učenja i učenja imitacijom [7]. Korištenje ML-Agents dopušta programerima već spomenuto stvaranje responzivnog i dojmljivog iskustva za korisnike.

Primjer korištenja Unityja i alata ML-Agents je treniranje agenta za igranje naprednih razina igre Snoopy Pop razvijene od strane tvrtke JamCity [8]. Snoopy Pop predstavlja igru pucanja mjehurića kako bi se oslobodio lik Woodstock i njegovo jato ptica. Igrač ispucava mjehuriće različitih boja pod raznim kutovima kako bi spojio tri ili više mjehurića iste boje i na taj način ih probušio i tako oslobodio ptice. Razina igre završava kada igrač uspješno oslobodi sve ptice, a gubi ako je ispucao sve mjehuriće koje ima, a nije uspio osloboditi sve ptice. Cilj treniranja ovog agenta bio je postići ponašanje igrača i pobijediti posljednju razinu igre. Koristeći alat ML-Agents za treniranje agenta postavljen je agent koji prima promatranja (engl. *observations*) i dobiva nagrade na temelju akcija koje izvršava. Zbog svih promatranja (engl. *observations*) koje prima agent i velikog broja akcija, treniranje agenta za rješavanje razina igre Snoopy Pop bilo je zahtjevno.

Budućnost strojnog učenja u industriji video igara je neizbježna i programeri će trebati nastaviti pronalaziti nove načine za njegovu implementaciju. Tako je i u ovom radu korišten Unity s alatom ML-Agents kako bi se stvorila željena okolina za treniranje agenta.

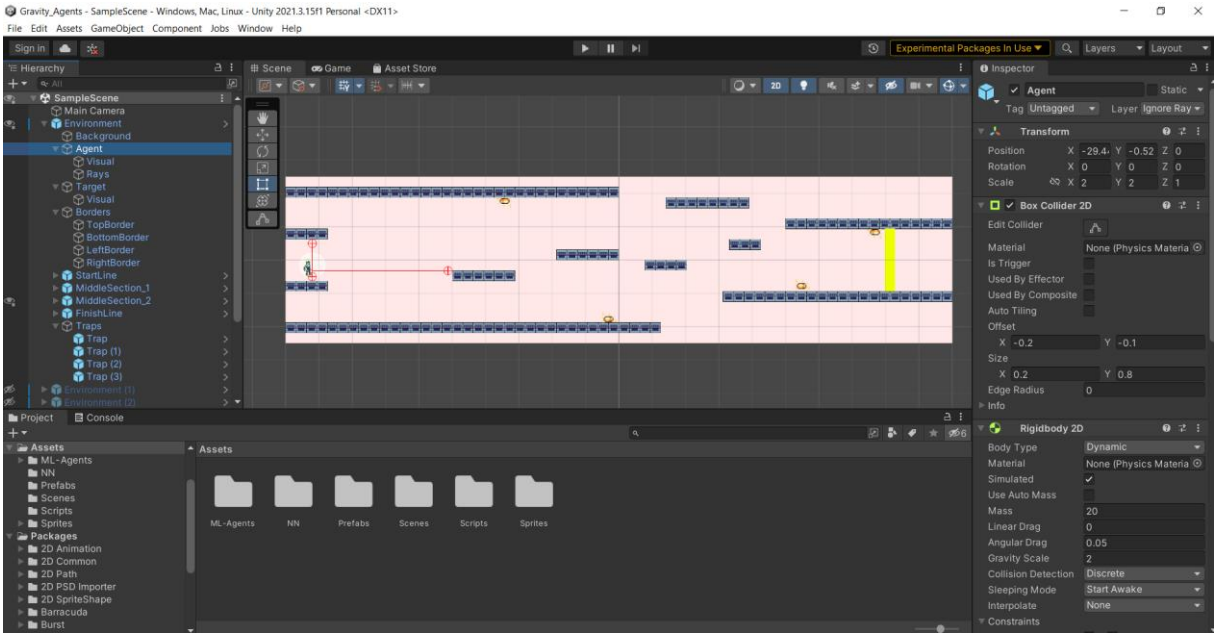
3. KORIŠTENE TEHNOLOGIJE I ALATI

Ovim poglavljem opisane su tehnologije i alati koji su bili potrebni pri razvoju računalne igre. Osim glavnih tehnologija poput Unityja za razvoj igre, Visual Studio Code uređivača koda i alata ML-Agents korišten je GIMP uređivač slika, TensorFlow platforma za strojno učenje i Anacoda za stvaranje okruženja za rad s Pythonom.

3.1. Unity

Unity [9], osnovan i razvijen 2005. godine od strane tvrtke Unity Technologies, predstavlja višeplatformski (engl. *cross-platform*) softver za razvoj igara. Osim što Unity podržava alate za razvoj 2D i 3D igara, on omogućuje programerima stvaranje sadržaja virtualne i proširene stvarnosti. Sve se to može implementirati na preko 25 različitih platformi [10] što olakšava posao programerima jer ne moraju mijenjati platforme [11]. Iako je Unity poznat po tome što pruža mogućnosti za razvoj video igara on se sve više koristi za 3D dizajn i simulaciju u filmskoj i automobilskoj industriji, ali i u arhitekturi [12]. Cilj Unityja [13] je i dalje pružanje najrobusnijeg skupa alata za razvoj igara s fokusom na jednostavnost korištenja softvera od strane programera što uključuje i početnike. Uz to, Unity ima mnogo dostupnih resursa za učenje što olakšava razvoj igara početnicima. Zaključno s time moglo bi se reći da se u Unityju može razviti igra bez pisanja koda koristeći razne značajke i dodatke, ali za kompleksnija ponašanja korištenih objekata potrebne su C# skripte. Uz ranije prednosti, Unity je također poznat kao istovremeno snažan i svestran softver. On nudi raznoliku ponudu dodatnih alata, a uz veliku ponudu sredstava za igre, osoba može stvoriti široki opseg rezultata. Na popularnost Unityja između ostalog utječe i njegova cijena. Unity je besplatan za sve korisnike do postignuća određene financijske dobiti od prodaje razvijenih igara ili drugih proizvoda. Veliku popularnost slijedi velika zajednica korisnika koji koriste Unity što potiče aktivan razvoj i ažuriranje postojećih značajki [13].

Unity grafičko sučelje ima mnogo različitih stavki koje se koriste za kreiranje cijele igre. Slika 3.1.1. predstavlja izgled Unity sučelja, a na njoj se može vidjeti glavni prozor (engl. *scene view*) koji prikazuje samo jednu scenu koja može biti razina igre, izbornik, okolina simulacije i slično. U tu scenu se postavljaju objekti i pomoću mnoštva ugrađenih značajki prilagođavaju njihove karakteristike kako bi se stekli željeni rezultati. Zbog velikih mogućnosti i prethodno navedene prednosti Unity je izabran za izradu platformske igre uz korištenje alata ML-Agents za strojno učenje.



Slika 3.1. Prikaz Unity sučelja

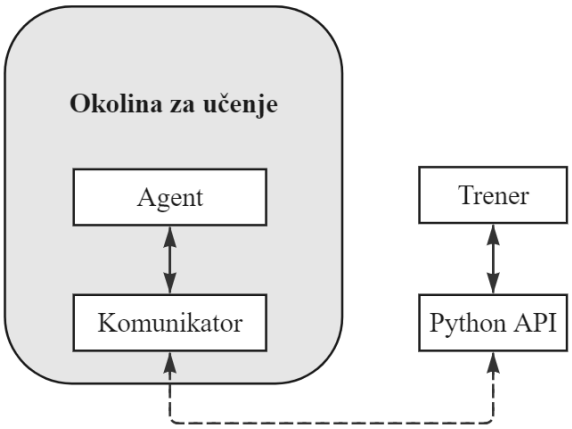
3.2. Unity Machine Learning Agents

ML-Agents, objavljen 2017. godine od strane tvrtke Unity [14], predstavlja alat koji omogućava kreiranje igara i simulacija koje služe kao okoline za treniranje inteligentnih agenata. Glavni cilj kreiranja alata ML-Agents bio je pružiti programerima i istraživačima umjetne inteligencije platformu za obuku i ugradnju inteligentnih agenata koristeći najnovija postignuća u strojnom učenju i najnaprednije algoritme za strojno učenje.

ML-Agents pruža dva algoritme za podržano učenje, a to su SAC (engl. *Soft Actor Critic*) i PPO (engl. *Proximal Policy Optimization*). SAC algoritam ima mogućnost učiti iz prijašnjih iskustava koja su spremljena u međuspremnik (engl. *buffer*) iz kojega se izvlače i koriste pri treniranju [15]. Glavna značajka SAC algoritma je regulacija entropije gdje se pokušava maksimizirati očekivani povrat i entropija, odnosno mjera slučajnosti [16]. Iz toga se može zaključiti da će agent postizati cilj zadatka djelujući nasumičnim akcijama. S druge strane je PPO algoritam koji predstavlja zadani algoritam i može se koristiti u okolinama s diskretnim i kontinuiranim akcijama. Cilj PPO algoritma je uspostaviti ravnotežu između važnih čimbenika kao što su jednostavnost implementacije, lakoća podešavanja parametara, složenost i učinkovitost uzorka uz što manja odstupanja od prethodnog ponašanja pri ažuriranju u svakom koraku [17].

Uz algoritme za podržano učenje alat ML-Agents pruža i dva algoritma za učenje imitacijom: GAIL (engl. *Generative Adversarial Imitation Learning*) i BC (engl. *Behavior Cloning*). GAIL algoritam [15] se može koristiti s ili bez nagrada u okolini. On najbolje radi s ograničenim brojem demonstracija. Uz neuronsku mrežu koja se koristi za treniranje agenta ovaj algoritam koristi dodatnu neuronsku mrežu za razlikovanje (engl. *discriminator*) demonstracija i treniranih ponašanja agenta. Uspoređujući naučeno i demonstrirano, neuronska mreža za razlikovanje dodjeljuje nagrade na temelju procjene prikladnosti ponašanja agenta, odnosno ponaša li se agent slično kao u demonstraciji. Svakim korakom treniranja agent pokušava ostvariti što veću nagradu sa što boljom imitacijom i paralelno tome neuronska mreža za razlikovanje postaje sve stroža u procjeni razlika između demonstracija i treniranog agenta. Za razliku od GAIL algoritma, BC algoritam [15] direktno trenira agenta da imitira akcije iz demonstracija. Za učenje imitacijom pomoću BC algoritma demonstracije trebaju imati sva potrebna stanja koja agent može doživjeti kako bi treniranje bilo zadovoljavajuće. BC algoritam može se koristiti pri treniranju sa SAC i PPO algoritmima ili u suradnji s GAIL algoritmom u slučaju nezadovoljavajućeg broja akcija u demonstraciji.

Alat ML-Agents sadrži komponente visoke razine među kojima su okolina za učenje, Python API, komunikator i Python treneri (engl. *trainers*) [15] što se može vidjeti na slici 3.2. Okolina za učenje obuhvaća scenu i sve objekte povezane s procesom učenja. Scena pruža okolinu u kojoj agenti izvršavaju akcije na temelju promatranja i tako uče. Komunikator se nalazi unutar okoline za učenje i povezuje ju s Python API-jem. Python API sadrži Python sučelje za interakciju i izmjenjivanje okoline za učenje. Python treneri obuhvaćaju sve algoritme za strojno učenje koji omogućuju treniranje agenata.

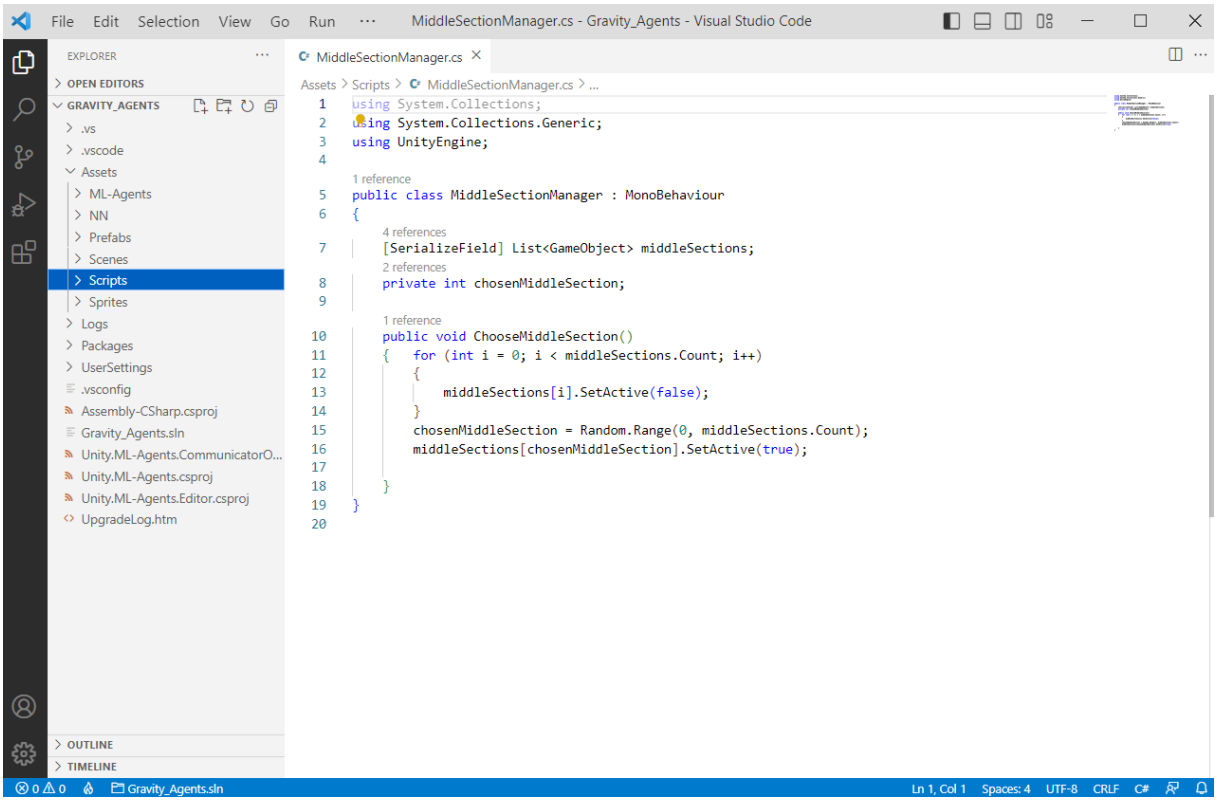


Slika 3.2. Komponente visoke razine alata ML-Agents

3.3. Visual Studio Code

Visual Studio Code, objavljen 2015. godine od strane tvrtke Microsoft [18], predstavlja moćan uređivač koda koji je dostupan za računala s Windows, macOS i Linux operacijskim sustavima. On omogućuje ugrađenu potporu za JavaScript, TypeScript i Node.js, a uz to je moguće koristiti i druge jezike preuzimajući proširenja za C++, C#, Java, Python, .NET i drugo [19]. Neke od značajki koje omogućuje Visual Studio Code su: pronalaženje i otklanjanje grešaka i nedostataka (engl. *debugging*), inteligentno dovršavanje koda, refaktoriranje koda, isticanje sintakse, ali također, korisnici mogu instalirati dodatna proširenja i na taj način dodati nove funkcionalnosti.

Visual Studio Code korišten je pri izradi skripti koje se koriste u Unityju za detaljnije definiranje funkcionalnosti korištenih objekata. Instalirana su i korištena proširenja za C# i .NET. Programski kod skripti pisan je objektno orijentiranim programskim jezikom C#. Slika 3.3. prikazuje snimku zaslona Visual Studio Code sučelja na kojoj se može vidjeti skripta za nasumično biranje srednjeg dijela razine za treniranje.



Slika 3.3. Prikaz Visual Studio Code sučelja

3.3.1. Programski jezik C#

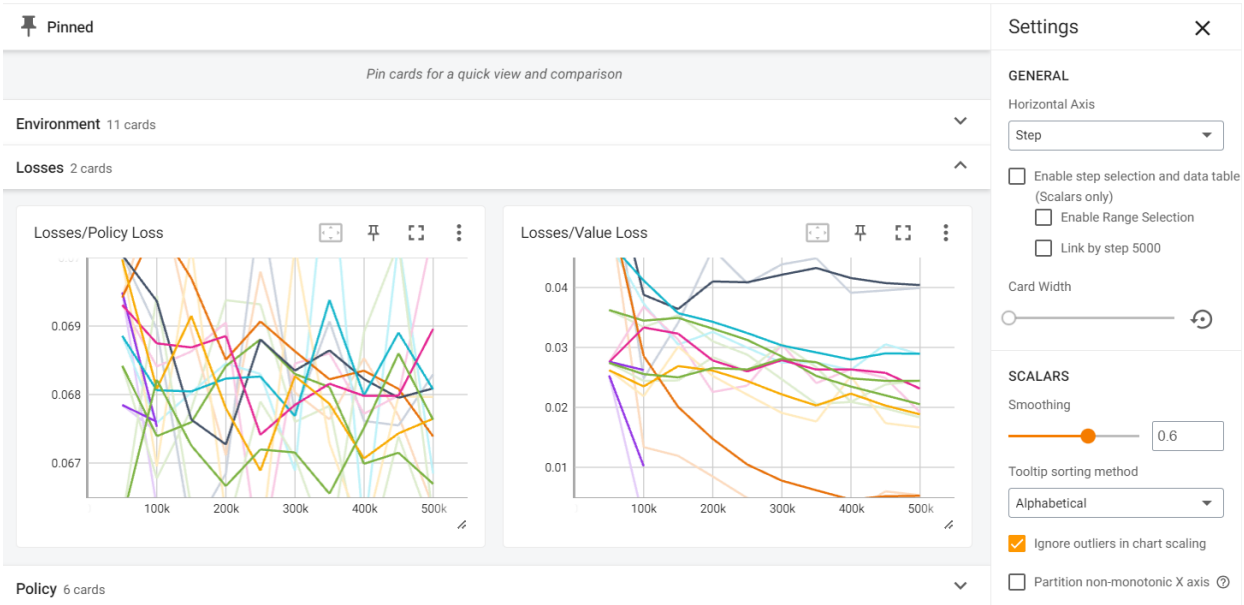
Programski jezik C# razvijen je unutar tvrtke Microsoft. Njegovu razvoju prvenstveno su pridonijeli Anders Hejlsberg, Scott Wiltamuth i Peter Golde. C# je objektno-orijentirani programski jezik koji je prvi put objavljen 2000. godine kako bi služio programerima za razvoj aplikacija baziranim na .NET sustavu. C# pripada porodici C programskih jezika jer dijeli karakteristike C i C++ programskih jezika, ali je također sličan Javi i JavaScriptu [20]. Neke od značajki koje su poboljšane u C# u odnosu na C i C++ su strogo definiran Boolean tip varijable i posjedovanje sakupljača smeća (engl. *garbage collector*), odnosno automatski se upravlja memorijom što sprječava curenje memorije i olakšava posao programera [21]. C# može se koristiti za razvoj mrežnih, računalnih i mobilnih aplikacija, ali i igara što se može vidjeti u ovom radu.

3.4. Dodatni alati

GNU Image Manipulation Program (GIMP), objavljen 1996. godine od strane studenata fakulteta Berkeley, Spencera Kimballa i Petera Mattisa [22], predstavlja program za uređivanje slika. GIMP je višeplatformski (engl. *cross-platform*) uređivač slika koji je dostupan za računala s Linux, Windows, macOS i drugim operacijskim sustavima [23]. Iako je GIMP besplatan on podržava različite alate za uređivanje i retuširanje slika, crtanje i slikanje, uređivanje i izmjenu fonta teksta. Za izvoz uređenih slika GIMP podržava formate PSD, JPG, GIF, PNG, TIFF, HEIF i WebP [24]. U ovom radu GIMP je korišten za grafičko manipuliranje modela objekata, odnosno mijenjanje boja korištenih 2D grafičkih objekata što se može vidjeti na slici 3.4. Ovi 2D grafički objekti (engl. *sprites*) kasnije su implementirani u Unityju.

Slika 3.4. Prikaz GIMP sučelja s uređenim objektom zamke

TensorFlow, objavljen 2015. godine od strane tvrtke Google [25], predstavlja biblioteku otvorenog koda za numeričko računanje i strojno učenje. TensorFlow sustav razvijen je s ciljem pružanja mogućnosti za eksperimentiranje s novim modelima, njihovo treniranje na velikoj količini podatak i na kraju njihovo korištenje u produkciji. On omogućuje strojno učenje koristeći veliki razmjer i raznovrsnost okolina. TensorFlow koristi dijagram toka podataka (engl. *dataflow graphs*) za predstavljanje izračuna, zajedničkih stanja i operacija za izmjenu tih stanja. Temelji se na arhitekturi koja preslikava čvorove iz dijagrama toka podataka na nakupine strojeva, čime daje fleksibilnost programeru za eksperimentiranje s novim optimizacijama i algoritmima za učenje [26]. TensorFlow podržava razne aplikacije, s fokusom na treniranje i zaključivanje o dubokim neuronskim mrežama. U ovom radu korišten je TensorBoard, paket za vizualizaciju TensorFlowa, za prikaz pohranjenih statistika tijekom treniranja pomoću alata ML-Agents što se može vidjeti na slici 3.5.



Slika 3.5. Prikaz TensorBoard sučelja sa statistikom treniranja pomoću alata ML-Agents

Anaconda, razvijena 2012. godine od strane Petera Wanga i Travisa Oliphanta [27], predstavlja upravitelj paketima za Python i R. Izvorno je razvijena kako bi riješila poteškoće kod upravljanja paketima pri radu s analizom podataka. Conda je višeplatformski (engl. *cross-platform*) paket i upravitelj okolinama koji instalira i upravlja Conda paketima iz repozitorija Anaconde [28]. Conda je podržana na macOS, Linux i Windows operacijskim sustavima, a za Windows se predlaže korištenje Anaconda Prompt naredbenog retka za pisanje naredbi [29]. U ovom radu Conda je korištena za upravljanje paketima i okolinom za treniranje agenta.

4. STROJNO UČENJE

Strojno učenje [30] predstavlja dio umjetne inteligencije koji se bavi korištenjem podataka i algoritama s ciljem razvoja sposobnosti računala za imitiranje inteligentnog ponašanja ljudi. „Zadatak algoritma strojnog učenja je pronaći prirodne uzorke i poveznice u podatcima te na temelju toga steći uvid i zatim odlučiti i predviđati.“ [4] Strojno učenje najbolje je primijeniti u slučaju kompleksnog zadatka s velikom količinom podataka i velikim brojem varijabli za koji ne postoje točno definirane formule ili jednadžbe. Tijek procesa strojnog učenja počinje s podatcima, a to mogu biti brojevi, tekst, slike ili bilo koja druga vrsta podataka. Potrebno je prikupiti i pripremiti podatke koji se zovu podatci za treniranje. Nakon toga programeri trebaju odabrati algoritme za strojno učenje i pokrenuti treniranje. Treniranje se temelji na pronalasku uzoraka na temelju kojih se stvaraju predviđanja. Tijekom treniranja programeri mogu promijeniti parametre korištenog algoritma kako bi se poboljšala točnost rezultata. Osim podataka za treniranje potrebno je izdvojiti i podatke koji će služiti za procjenu uspješnosti istreniranog modela, odnosno donosi li istrenirani model dobre rezultate u slučaju korištenja novih podataka. Time se dobiva model koji ispravno radi s različitim skupinama podataka, a ne samo podatcima koji su korišteni pri treniranju. Strojno učenje se dijeli na tri glavne kategorije, a to su nadzirano učenje, nenadzirano učenje i podržano učenje.

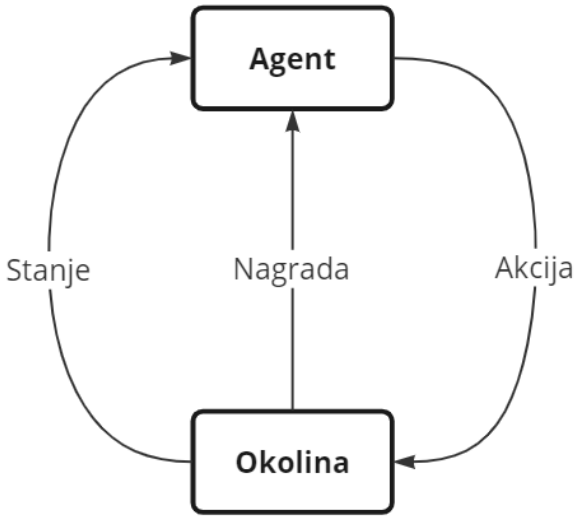
4.1. Nadzirano i nenadzirano učenje

Nadzirano učenje temelji se na razvoju modela koji se treniraju primjenom poznatih skupova ulaznih i izlaznih podataka. Znajući ulazne i izlazne skupove podataka trenira se model za predviđanje [4]. Algoritmi nadziranog učenja prikladni su za rješavanje zadataka binarne klasifikacije u kojima se podatci razdvajaju na dvije klase, više-klasne klasifikacije u kojima se podatci razvrstavaju na više od dvije klase. Uz klasifikaciju koja se koristi za predviđanje diskretnih odziva, za predviđanje kontinuiranih odziva koristi se regresija.

Za razliku od nadziranog učenja kod nenadziranog učenja poznat je samo ulazni skup podataka. Nenadzirano učenje temelji se na pronalasku skrivenih uzoraka koji se mogu koristiti za sortiranje podataka. Algoritmi nenadziranog učenja najčešće se koriste za rješavanje zadataka grupiranjem gdje se skup podataka dijeli u grupe na temelju njihovih sličnosti. Uz grupiranje, algoritmi nenadziranog učenja pogodni su za rješavanje zadataka otkrivanja anomalija u kojima se identificiraju neobični podatci u skupu podataka, asocijacijsko rudarenje u kojima se identificiraju skupovi stavki u skupu podataka koje se često zajedno pojavljuju i za smanjenje dimenzionalnosti gdje se smanjuje broj varijabli u skupu podataka [30].

4.2. Podržano učenje

Podržano učenje funkcionira na principu pokušaja i pogrešaka gdje se agent trenira da poduzme najbolju akciju za koju dobiva nagradu [30]. Algoritam za podržano učenje sadrži jasne ciljeve i propisan skup pravila za postizanje tih ciljeva, a agent teži dobivanju pozitivnih nagrada i izbjegavanju akcija koje rezultiraju negativnim nagradama, odnosno kaznama. Ovaj algoritam se koristi u području robotike, gdje robot može učiti akcije u virtualnoj okolini i onda ih primijeniti u stvarnom svijetu. Također, podržano učenje je prigodno za upravljanje resursima što poduzećima pojednostavljuje planiranje alokacije dostupne, poznate i ograničene količine resursa ovisno o definiranom skupu krajnjih ciljeva [31]. Između ostalog, algoritmi podržanog učenja prikladni su i u području video igara što je prikazano u ovom radu. Ciklus podržanog učenja sastoji se od agenta koji izvršava akcije u okolini na temelju kojih mijenja stanje i dobiva nagradu što se može vidjeti na slici 4.1.



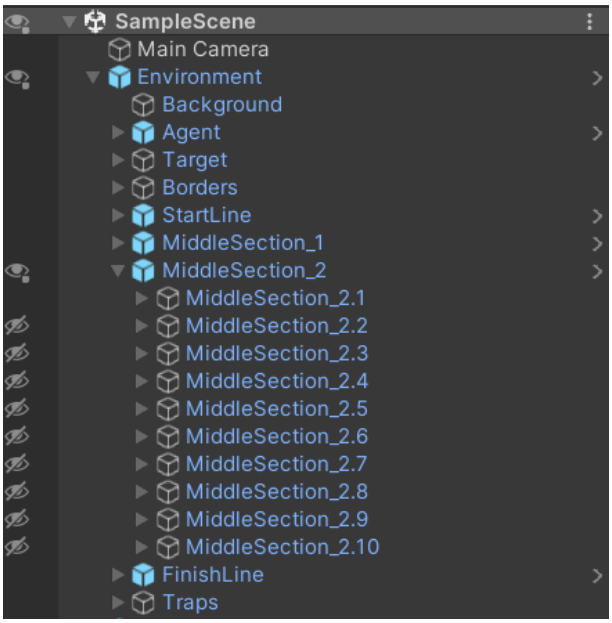
Slika 4.1. Ciklus podržanog učenja

5. IZRADA PROJEKTA

Ovo poglavlje se sastoji od opisa izrade okoline za treniranje, zajedno s agentom te algoritama korištenih za treniranje. Osim okoline za treniranje, opisana je izrada okoline za testiranje treniranog agenta te su razjašnjeni rezultati treninga. U konačnici objašnjeno je korištenje i implementacija treniranog agenta te kako se on uklapa u cjelokupni projekt Gravity Agents. Ovaj projekt će pokazati sposobnosti ML-Agents alata u Unityju za stvaranje i treniranje agenta za rješavanje različitih razina platformske igre rabeći odabrane algoritme strojnog učenja.

5.1. Izrada jednostavne okoline za treniranje agenta

Za početak kreirana je jednostavna okolina za treniranje agenta. Ova okolina sastoji se od početnog odjeljka, prvog srednjeg odjeljka, nasumično izabranog drugog srednjeg odjeljka i krajnjeg odjeljka što se može vidjeti na slici 5.1. Uz odjeljke platformi u hijerarhiji objekata nalazi se i objekti pozadine, cilja, granica i zamki te sam agent. Početni odjeljak platformi, prvi srednji odjeljak i krajnjeg odjeljak su jednaki u svim scenama.



Slika 5.1. Prikaz hijerarhija objekata okoline

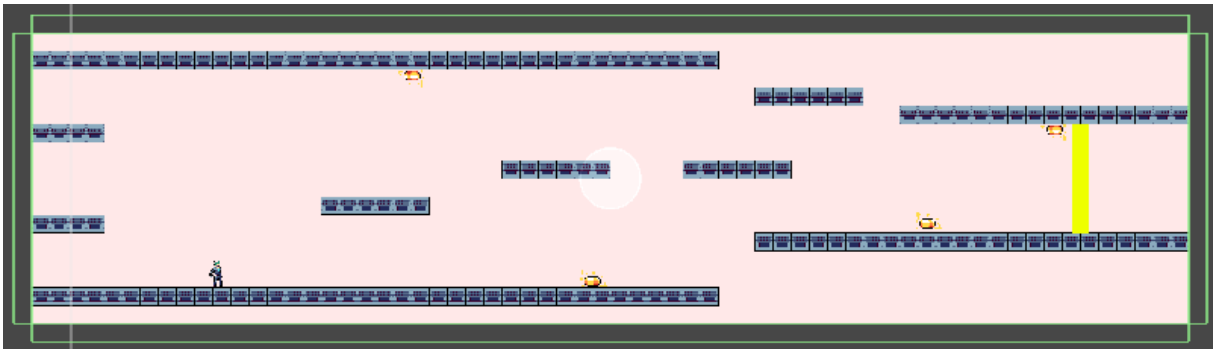
Prema slici 5.1. primjećuje se postojanje 10 srednjih odjeljaka. Ti odjeljci se nasumično biraju pomoću skripte *MiddleSectionManager* koristeći funkciju *ChooseMiddleSection* koja se poziva na početku kreiranja epizode treniranja. U funkciji *ChooseMiddleSection* nasumično se odabire broj između nule i broja odjeljaka, odnosno 10, koristeći funkciju *Range*. Pomoću funkcije *SetActive* aktivira se odabrani odjeljak iz liste odjeljaka (Programski kod 5.1.).

Linija ***Kod***

```
1:      chosenMiddleSection = Random.Range(0, middleSections.Count);
2:      middleSections[chosenMiddleSection].SetActive(true);
```

Programski kod 5.1. Isječak koda iz funkcije ChooseMiddleSection koji nasumično odabire odjeljak

Svaki srednji odjeljak je različit, čime se dobiva 10 različitih okolina u kojima se provodi treniranje agenta. Na slici 5.2. prikazan je izabrani prvi objekt liste drugog srednjeg odjeljka. Ostali objekti bit će prikazani u nastavku pri treniranju agenta gdje će se paralelno koristiti više okolina.

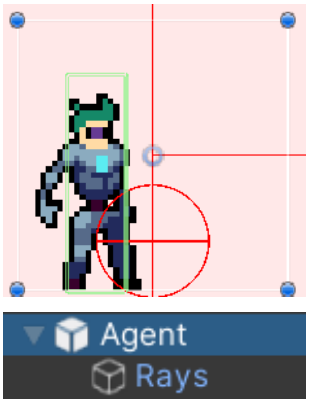


Slika 5.2. Prikaz izgleda okoline

Svim objektima odjeljaka, a tako i objektu cilja, objektima granica i objektima zamki dodijeljena je ugrađena komponenta Unityja pod nazivom *Box Collider 2D.* Ova komponenta koristi se za detektiranje sudara među objektima. Također, svim prethodno navedenim objektima pridružena je pripadajuća oznaka (engl. *tag*) i označena je kućica svojstva *IsTrigger* kako bi se kasnije u kodu moglo reagirati na sudare agenta s ovim objektima. Osim maske sloja (engl. *layermask*), objektima platformi dodijeljena je i pripadajuća oznaka *Platform*, objektu cilja oznaka *Target*, objektima granica oznaka *Border*, objektima zamki oznaka *Trap*. Kako bi se treniranje moglo sastojati od više od jedne okoline, položaji svih objekata okoline su lokalni. Korištenje više okolina bit će demonstrirano pri treniranju agenta.

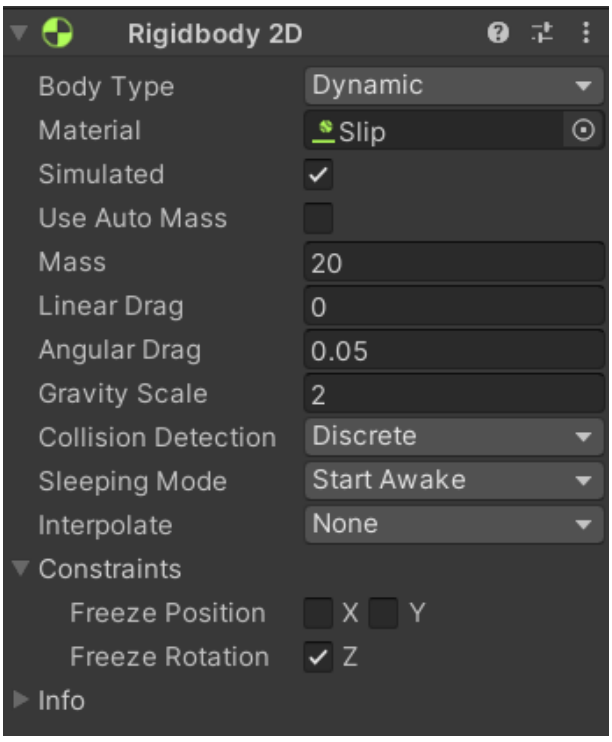
5.2. Stvaranje agenta

Agent predstavlja temeljni dio okoline učenja koji se trenira kako bi postigao željeno ponašanje. U ovom projektu cilj je istrenirati agenta za uspješno rješavanje različitih razina platformske igre. Agent objekt je jedan *GameObject* koji sadrži jedan dijete objekt *Rays.* Objekt *Rays* predstavlja senzor percepcije zraka.



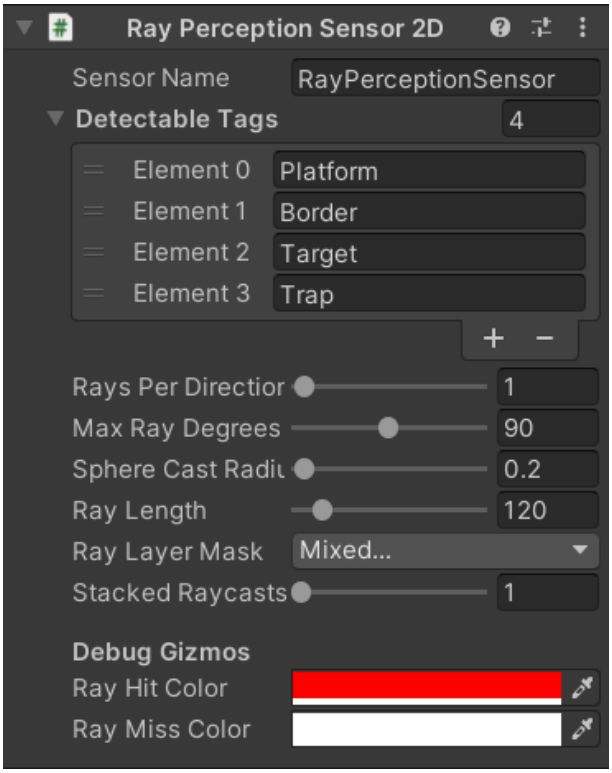
Slika 5.3. Prikaz izgleda i hijerarhije agenta

Na slici 5.3. vidljiv je zeleni pravokutnik koji predstavlja komponentu *Box Collider 2D* koja se koristi za detektiranje sudara s drugim objektima u sceni. Uz komponentu *Box Collider 2D* agentu je pridružena komponenta *Rigidbody 2D* koja omogućuje objektu reagiranje na fizičke promjene, među kojima je promjena gravitacije. U komponenti *Rigidbody 2D* postavljeno je svojstvo materijal (engl. *material*) na *Slip* što predstavlja stvoreni materijal s trenjem jednakim nuli, svojstvo mase (engl. *mass*) na 20, svojstvo gravitacijske ljestvice (engl. *gravity scale*) na vrijednost dva i označena je kućica svojstva zamrznuti rotaciju (engl. *freeze rotation*) kako se agent ne bi rotirao po z osi. Ostala svojstva ostavljena su sa zadanim vrijednostima što se vidi na slici 5.4.



Slika 5.4. Komponenta Rigidbody 2D sa svojstvima

Crvene zrake vidljive na slici 5.3. predstavljaju zrake senzora percepcije, a dio su komponente *Ray Perception Sensor 2D*. Ova komponenta pridružena je objektu *Rays* pod imenom *RayPerceptionSensor*. Ona automatski dodaje promatranja (engl. *observations*) agentu čime uklanja potrebu za definiranjem zraka percepcije u kodu u funkciji *CollectObservations*. Svojstvo oznaka koje se mogu otkriti (engl. *detectable tags*) postavljeno je na vrijednost četiri i uključuje oznake za platforme, granice, cilj i zamke. Svojstvo zraka po smjeru (engl. *rays per direction)* određuje broj zraka s bilo koje strane centralne linije. Ovo svojstvo ima vrijednost jedan što znači da postoje tri zrake. Kut između zraka definiran je svojstvom maksimalnih stupnjeva zraka (engl. *max ray degrees*), gdje vrijednost 90 stvara prave kutove između zraka. Ova vrijednost je izabrana kako bi agent mogao zaključiti iz promatranja što se nalazi ispred, iznad i ispod njega. Veličina kružnice na kraju zrake definirana je svojstvom polumjera kugle bacanja (engl. *sphere cast radius*) i ima vrijednost 0.2. Vrijednost svojstva dužine zrake (engl. *ray length*) je 120 kako bi agent ima mogućnost opaziti cilj razine. Ostala svojstva imaju zadane vrijednosti što je vidljivo na slici 5.5.



Slika 5.5. Komponenta Ray Perception Sensor 2D sa svojstvima

Agentu je pridružena skripta *GravityAgent* koja kontrolira agenta i nasljeđuje klasu *Agent*, ali i dva sučelja *IScenes* i *IGravity*. *IScenes* sučelje sadrži *enum Name*, odnosno klasu konstanti, u kojoj se nalaze nepromjenjive varijable naziva scena (Programski kod 5.2.). Ovi nazivi koriste se pri dohvaćanju imena scena umjesto korištenja *string* varijabli.

Linija ***Kod***

```
1:      public interface IScenes
2:      {
3:          enum Name{
4:              MainMenu,
5:              Level_1,
6:              Level_2,
7:              Level_3,
8:              TrainingLevel,
9:              SampleScene
10:         };
11:     }
```

Programski kod 5.2. Sučelje IScenes

Sučelje *IGravity* osigurava implementaciju funkcija koje su potrebne pri promijeni gravitacije. Među tim funkcijama su funkcije *IsGrounded, GravitySwitch, Rotate, Flip* i *Spawn.* Sve funkcije implementirane su u klasi *GravityAgent.* Funkcija *IsGrounded* koristi se za provjeru nalazi li se agent na platformi ili je u zraku. Agent se nalazi u zraku pri postupku promjene gravitacije i u tom trenutku se ne bi trebao moći pomicati lijevo ili desno. Provjera položaja agenta postignuta je primjenom funkcije *BoxCast* na objektu klase *Physics2D* koja vraća objekta klase *RaycastHit2D* s referencom na objekt s kojim se sudara. U funkciji je moguće definirati masku sloja (engl. *layermask*) kako bi se selektivno detektirali objekti određenog sloja. Funkcijom *BoxCast* provodi se detekcija samo na sloju platformi te u slučaju da ne dođe do detekcije niti jednog objekta vraća *null*. Također, potrebno je provjeriti gravitaciju te ako je došlo do promjene koristi se obrnuti vektor jednake duljine (Programski kod 5.3.).

Linija ***Kod***

```
1:      public bool IsGrounded()
2:      {
            RaycastHit2D raycastHit =
        Physics2D.BoxCast(boxCollider.bounds.center,
3:      boxCollider.bounds.size, 0f, Vector2.down, 0.4f, platformLayer);
4:          if (top == true)
5:          {
                raycastHit = Physics2D.BoxCast(boxCollider.bounds.center,
6:      boxCollider.bounds.size, 0f, Vector2.up, 0.4f, platformLayer);
7:          }
8:          return raycastHit.collider != null;
9:      }
```

Programski kod 5.3. Isječak funkcije IsGrounded iz klase GravityAgent

Funkcija *GravitySwitch* mijenja gravitaciju agenta i poziva funkcije *Rotate* i *Flip*. Obje funkcije *Rotate* i *Flip* rotiraju trodimenzionalnom rotacijom pomoću svojstva *eulerAngles*. Razlika između tih funkcija je smjer rotacija, u funkciji *Rotate* agent se rotira oko z osi, odnosno gore-dolje, a u funkciji *Flip* agent se oko y osi, odnosno lijevo-desno. Ove funkcije sukladno promjenama mijenjaju vrijednosti varijabli *top* i *facingRight* koje definiraju rotaciju agenta (Programski kod 5.4.).

Linija ***Kod***

```
1:      public void GravitySwitch()
2:      {
3:          rigidbody.gravityScale *= -1;
4:          Rotate();
5:          facingRight = !facingRight;
6:          Flip();
7:      }
8:      public void Rotate()
9:      {
            transform.localEulerAngles = top ? Vector3.zero : new Vector3(0,
10:     0, 180f);
            top = !top;
11:     }
12:     public void Flip()
13:     {
14:         facingRight = !facingRight;
            if (top == false)
15:         {
                transform.localEulerAngles = facingRight ? Vector3.zero :
16:     new Vector3(0, 180f, 0);
17:         }
18:         else
19:         {
                transform.localEulerAngles = facingRight ? new Vector3(0,
20:     180f, 180f) : new Vector3(0, 0, 180f);
21:         }
22:     }
```

Programski kod 5.4. Isječak funkcija GravitySwitch, Rotate i Flip iz klase GravityAgent

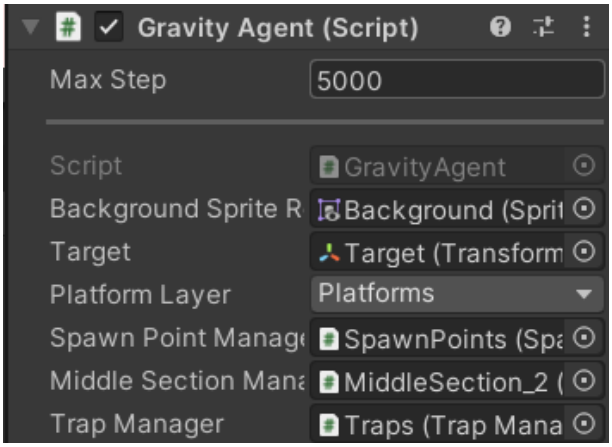
Funkcija *Spawn* poziva se pri početku epizode treninga te osigurava ispravnost gravitacije agenta u slučaju da je razina završila s promjenom gravitacije. Ako je gravitacija agenta negativna, što znači da je gravitacija promijenjena, ova funkcija poziva funkciju *GravitySwitch* kako bi se ispravila gravitacija agenta (Programski kod 5.5.).

Linija ***Kod***

```
1:      public void Spawn()
2:      {
3:          if (rigidbody.gravityScale < 0)
4:          {
5:              GravitySwitch();
6:          }
7:      }
```

Programski kod 5.5. Isječak funkcije Spawn iz klase GravityAgent

Klasa *Agent* koju nasljeđuje klasa *GravityAgent* sadrži funkcionalnosti za strojno učenje i treniranje agenta. Od tih funkcija u klasi *GravityAgent* prepisane su funkcije *OnEpisodeBegin*, *CollectObservations*, *OnActionReceived* i *Heuristic*. Osim funkcija, skripti *GravityAgent* u uređivaču se dodjeljuje varijabla maksimalnog broja koraka (engl. *max step*) koje agent ostvaruje. Za jednostavnu okolinu primjereni maksimalni broj koraka je 5000. Taj broj je promijenjen za naprednu okolinu u 25000. Skripti se predaju odgovarajući objekti što se može vidjeti na slici 5.6.



Slika 5.6. GravityAgent skripta u Unity uređivaču

Funkcijom *OnEpisodeBegin* resetira se okolina učenja, a poziva se na početku agentove epizode učenja (Programski kod 5.6.). Tom funkcijom inicijalizirani su objekti okoline učenja. Ovisno o sceni u kojoj se kod izvodi, pozivaju se odgovarajuće funkcije. Kako bi se provjerilo ime scene, potrebno je pozvati funkciju *GetActiveScene*().*name* na objektu klase *SceneManager*. Za postavljanje zamki za jednostavnu okolinu poziva se funkcija *SetTrapsTraining* na objektu klase *TrapManager*, a za naprednu funkcija *SetTrapLevel*. Klasi *TrapManager* predaje se lista zamki koje se razlikuju samo po položaju u okolini. Položaji svih zamki i ciljeva su lokalni, ali su nasumično odabrani kako bi se mogli koristiti u više okolina (Programski kod 5.7.).

Linija Kod

```
1:      using UnityEngine;
2:      using Unity.MLAgents;
3:      using Unity.MLAgents.Actuators;
4:      using Unity.MLAgents.Sensors;
5:      using UnityEngine.SceneManagement;
6:
7:      public class GravityAgent : Agent, IScenes, IGravity {
            [SerializeField] private SpriteRenderer
8:      backgroundSpriteRenderer;
9:          [SerializeField] private Transform target;
10:         [SerializeField] private LayerMask platformLayer;
11:         [SerializeField] private SpawnPointManager spawnPointManager;
            [SerializeField] private MiddleSectionManager
12:     middleSectionManager;
13:         [SerializeField] private TrapManager trapManager;
14:         private new Rigidbody2D rigidbody;
15:         private BoxCollider2D boxCollider;
16:         private bool top = false;
17:         private bool facingRight = true;
18:
19:         public override void OnEpisodeBegin(){
20:             rigidbody = GetComponent<Rigidbody2D>();
21:             boxCollider = GetComponent<BoxCollider2D>();
                if (SceneManager.GetActiveScene().name ==
22:     IScenes.Name.SampleScene.ToString())
23:             {
24:                 trapManager.SetTrapsTraining();
                    target.localPosition = new Vector3(Random.Range(26, 31),
25:     0);
26:             }
27:             else
28:             {
29:                 trapManager.SetTrapsLevel();
                    target.localPosition = new Vector3(Random.Range(202,
30:     207), 0);
31:             }
32:             middleSectionManager.ChooseMiddleSection();
33:             Spawn();
                transform.localPosition =
34:     spawnPointManager.ChooseSpawnPoint();
35:         }
```

Programski kod 5.6. Isječak koda iz klase GravityAgent s klasom OnEpisodeBegin

Linija Kod

```
1:      using System.Collections.Generic;
2:      using UnityEngine;
3:      public class TrapManager : MonoBehaviour
4:      {
5:          [SerializeField] private List<Transform> traps;

6:          public void SetTrapsTraining()
7:          {
                traps[0].localPosition = new Vector3(Random.Range(-9f, -
8:      1.5f), -5.5f);
                traps[1].localPosition = new Vector3(Random.Range(-15f, -
        10f), 5.5f);
                traps[2].localPosition = new Vector3(Random.Range(17.5f,
9:      20f), -2.5f);
                traps[3].localPosition = new Vector3(Random.Range(21.5f,
10:     24f), 2.5f);
11:         }
12:         public void SetTrapsLevel()
13:         {
                traps[0].localPosition = new Vector3(Random.Range(-9f, -
14:     1.5f), -5.5f);
                traps[1].localPosition = new Vector3(Random.Range(-15f, -
15:     10f), 5.5f);
                traps[2].localPosition = new Vector3(Random.Range(193.5f,
16:     196f), -2.5f);
                traps[3].localPosition = new Vector3(Random.Range(197.5f,
17:     200f), 2.5f);
18:         }
19:     }
```

Programski kod 5.7. Klasa TrapManager

Kao što položaj zamki ovisi o okolini tako ovisi i položaj cilja koji se postavlja nasumično u određenom rasponu. Neovisno radi li se o jednostavnoj ili naprednoj okolini, na objektu klase *MiddleSectionManager* poziva se funkcija *ChooseMiddleSection*. Ova funkcija je opisana u prethodnom poglavlju, a za naprednu okolinu će biti opisana naknadno. Također, u funkciji *OnEpisodeBegin* poziva se već spomenuta funkcija *Spawn.*

Položaj agenta na početku epizode jednak je odgovoru funkcije *ChooseSpawnPoint* na objektu klase *SpawnPointManager*. Iz liste mjesta pojavljivanja (engl. *spawn point*) agenta nasumično se odabire jedan vektor položaja (Programski kod 5.8.). Time se omogućuje da agent ne započinje treniranje uvijek na istom položaju već se njegov položaj nasumično mijenja.

Linija ***Kod***

```
1:      using System.Collections.Generic;
2:      using UnityEngine;
3:
4:      public class SpawnPointManager : MonoBehaviour
5:      {
6:          [SerializeField] private List<Transform> spawnPoints;
7:          public Vector3 ChooseSpawnPoint()
8:          {
                return spawnPoints[Random.Range(0,
9:      spawnPoints.Count)].localPosition;
10:         }
11:     }
```

Programski kod 5.8. Klasa SpawnPointManager

Za prikupljanje promatranja (engl. *observations*) prepisana je funkcija *CollectObservations*. Uz prethodno spomenuta promatranja koja se automatski dodaju agentu, u ovoj funkciji još se pridodaju promatranja položaja agenta i cilja. Promatranja se dodaju u obliku dvodimenzionalnih vektora jer lokalni položaji promatranih objekata ne ovise o z osi (Programski kod 5.9.).

Linija ***Kod***

```
1:      public override void CollectObservations(VectorSensor sensor)
2:      {
3:          sensor.AddObservation((Vector2)transform.localPosition);
4:          sensor.AddObservation((Vector2)target.localPosition)
5:      }
```

Programski kod 5.9. Isječak funkcije CollectObservations iz klase GravityAgent

Prepisana funkcija *OnActionReceived* definira ponašanje agenta tijekom treniranja, odnosno akcije koje agent izvršava (Programski kod 5.10.). Ova funkcija provodi se nakon svake diskretne akcije agenta. Razlikuju se dvije diskretne akcije čije se vrijednosti predaju varijablama *moveX* i *changeGravity*. Varijabla *moveX* može poprimiti diskretne vrijednosti nula, jedan ili dva, gdje nula predstavlja odsutnost pomicanja lijevo-desno, jedan kretanje ulijevo, dva kretanje udesno. Pri izvršavanju ove akcije provjerava se položaj agenta i ako je potrebno mijenja se njegov smjer pozivom funkcije *Flip*. Ovisno o smjeru kretanja agent postiže nagrade. *AddReward* je funkcija koju podržava ML-Agents alat i koristi se za mijenjanje osnovne nagrade čija je vrijednost nula. Njeno korištenje utječe na sveukupnu vrijednost nagrade. Mala pozitivna nagrada se dodjeljuje u slučaju kada se agent pomiče prema naprijed, bliže cilju, a jednaka negativna ako se pomiče u

suprotnom smjeru. Ovime se potiče agenta na pravilno ponašanje, kretanje prema cilju, te izbjegava kontinuirano kretanje lijevo-desno na jednom mjestu. Vrijednost druge diskretne akcije predaje se varijabli *changeGravity*. Ova akcija se izvršava ako je vrijednost varijable *moveX* jednaka nuli. Ako je vrijednost varijable *changeGravity* jednaka nuli ne mijenja se gravitacija agenta, a ako je jednaka jedan onda se gravitacija mijenja pozivom funkcije *GravitySwitch*. Izvršavanjem ove akcije agent ne dobiva nagradu tako da sveukupna nagrada ne ovisi o broju promjena gravitacije u procesu postizanja cilja.

Linija ***Kod***

```
1:      public override void OnActionReceived(ActionBuffers actions)
2:      {
3:          float moveX = actions.DiscreteActions[0];
4:          float changeGravity = actions.DiscreteActions[1];
5:          float speed = 5f;
6:
7:          if (moveX == 1)
8:          {
9:              AddReward(-0.01f);
10:             if (facingRight)
11:             {
12:                 Flip();
13:             }
                transform.localPosition += Vector3.left * speed *
14:     Time.deltaTime;
15:         }
16:         else if (moveX == 2)
17:         {
18:             AddReward(0.01f);
19:             if (!facingRight)
20:             {
21:                 Flip();
22:             }
                transform.localPosition += Vector3.right * speed *
23:     Time.deltaTime;
24:         }
25:         else if (changeGravity == 1)
26:         {
27:             GravitySwitch();
28:         }
29:     }
```

Programski kod 5.10. Isječak funkcije OnActionReceived iz klase GravityAgent

Zadnja prepisana funkcija je *Heuristic*, ona omogućuje ručno upravljanje agentom. Ručnim upravljanjem agenta testira se okolina prije postupka treniranja kako bi se potvrdila njena ispravnost. U ovoj funkciji koristi se podatkovna struktura *ActionSegmet* koja omogućuje pristup segmentu osnovnog niza bez potrebe kopiranja i dodjela podnizova. U tu strukturu pohranjuju se vrijednosti diskretnih akcija. Ovisno o odabiru definiranih tipki na tipkovnici uzimaju se diskretne vrijednosti od nula do dva (Programski kod 5.11.).

Linija ***Kod***

```
1:      public override void Heuristic(in ActionBuffers actionsOut)
2:      {
3:          ActionSegment<int> discreteActions = actionsOut.DiscreteActions;
4:          if (IsGrounded())
5:          {
6:              if (Input.GetKey(KeyCode.A))
7:              {
8:                  discreteActions[0] = 1;
9:              }
10:             if (Input.GetKey(KeyCode.D))
11:             {
12:                 discreteActions[0] = 2;
13:             }
14:             discreteActions[1] = Input.GetKey(KeyCode.W) ? 1 : 0;
15:         }
16:     }
```

Programski kod 5.11. Isječak funkcije Heuristic iz klase GravityAgent

Uz prethodno opisane prepisane funkcije klase *Agent* potrebno je definirati funkciju kojom se određuje kraj epizode. Ova funkcija poziva se pri detekciji sudara agenta s okidačem (engl. *trigger*). Okidači su prethodno navedeni objekti koji imaju oznake *Target*, *Border* i *Trap*. Pri detekciji sudara s objektom provjerava se njegova oznaka te se na temelju nje dodjeljuje nagrada i završava epizoda. S obzirom na to da se ovaj agent koristi i nakon treniranja potrebno je provjeriti naziv scene. Ako sudareni objekt ima oznaku *Target* i koristi se u sceni treniranja, dodjeljuje mu se pozitivna nagrada plus dva, mijenja se boja pozadine u zelenu i završava epizoda. S druge strane, ako se agent koristi nakon treniranja također se završava epizoda, ali se scena ponovno učitava jer to znači da je agent došao do cilja prije igrača. U slučaju sudara s objektom koji ima oznake *Border* ili *Trap* i koristi se u sceni treniranja, agentu se dodjeljuje negativna nagrada minus dva i mijenja se boja pozadine u crvenu. Završetak epizode u ovom slučaju ne ovisi o nazivu scene (Programski kod 5.12.).

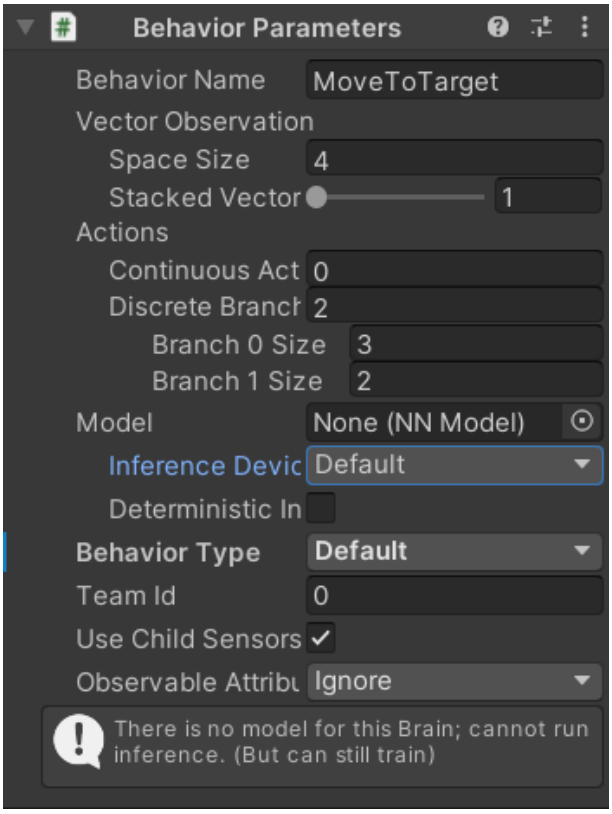
Linija ***Kod***

```
1:      private void OnTriggerEnter2D(Collider2D collider)
2:      {
3:          if (collider.gameObject.CompareTag("Target"))
4:          {
                if (SceneManager.GetActiveScene().name ==
        IScenes.Name.SampleScene.ToString() ||
        SceneManager.GetActiveScene().name ==
5:      IScenes.Name.TrainingLevel.ToString())
6:              {
7:                  AddReward(2f);
8:                  backgroundSpriteRenderer.color = Color.green;
9:                  EndEpisode();
10:             }
11:             else
12:             {
13:                 EndEpisode();
14:     SceneManager.LoadScene(SceneManager.GetActiveScene().name);
15:             }
16:         }
            if (collider.gameObject.CompareTag("Border") ||
17:     collider.gameObject.CompareTag("Trap"))
18:         {
                if (SceneManager.GetActiveScene().name ==
        IScenes.Name.SampleScene.ToString() ||
        SceneManager.GetActiveScene().name ==
19:     IScenes.Name.TrainingLevel.ToString())
20:             {
21:                 AddReward(-2f);
22:                 backgroundSpriteRenderer.color = Color.red;
23:             }
24:             EndEpisode();
25:         }
26:     }
```

Programski kod 5.12. Isječak funkcije OnTriggerEnter2Diz klase GravityAgent

Uz opisanu skriptu *GravityAgent* dodana je komponenta *Behaviour Parameters* (Slika 5.7.). Unutar komponente svojstvo ime ponašanja (engl. *behavior name*) postavljeno je na *MoveToTarget* što određuje ime modela koji će se stvoriti treniranjem. Svojstvo vektorsko promatranje (engl. *vector observation*) obuhvaća podatke promatranja prikupljene funkcijom *CollectObservations* i veličinu prostora (engl. *space size*) vrijednosti četiri jer koristi podatke dva dvodimenzionalna vektora. Svojstvom akcije (engl. *actions*) definiran je broj grana diskretnih akcija (engl. *discrete branches*) vrijednosti dva, gdje je veličina prve grane jednaka tri, a druge dva. Nakon treniranja svojstvu model pridružuje se prikladni istrenirani model neuronske mreže.

Način ponašanja agenta određen je svojstvom tipa ponašanja (engl. *behaviour type*). Ovo svojstvo se bira između tri opcije. Opcija *Default* se koristi tijekom treniranja, *Heuristic Only* za ručno upravljanje agentom i *Inference Only* za ponašanje agenta prema istreniranom modelu. Ostale opcije ostavljene su prema zadanim vrijednostima.



Slika 5.7. Komponenta Behaviour Parameters sa svojstvima

5.3. Treniranje agenta u jednostavnoj okolini

Kako bi treniranje agenta bilo učinkovitije korišteno je 12 okolina u sceni. Za pregled tih okolina korištena je glavna kamera sa zadanim svojstvima, ali promijenjena joj je veličina kako bi se vidjele sve okoline. Za treniranje agenta korišteni su prethodno spomenuti algoritmi PPO i SAC. Nakon njihova izvođenja uspoređeni su njihovi rezultati i odabran je bolji za treniranje napredne okoline. Oba algoritma zahtijevaju postavljanje hiperparametara treniranja koji se nalaze u yaml datoteci. Ova datoteka se referencira pri pokretanju treniranja.

Zajednički skup parametara za treniranje oba algoritma sastoji se od sljedećih parametara: *trainer_type*, *summary_freq*, *time_horizon*, *max_steps*, *keep_checkpoints* [32]. Parametar *trainer_type* određuje tip algoritma korišten za treniranje. On je postavljen na vrijednost *ppo* za PPO algoritam i *sac* za SAC algoritam. Vrijednosti ostalih prethodno navedenih parametara su

ostavljene na zadane vrijednosti za oba algoritma. *Summary_freq* parametar određuje broj iskustava koje je potrebno prikupiti prije stvaranja i prikazivanja statistika treniranja. Zrnatost ili granuliranost grafa u TensorBoardu ovisi o ovom parametru jer je njime određena razina detaljnosti podataka. Parametar *time_horizon* predstavlja broj koraka iskustva koje je potrebno prikupiti po agentu prije njihovog dodavanja u međuspremnik iskustava. Ako se granica dosegne prije kraja epizode, procijenjena se vrijednost koristi za predviđanje ukupne očekivane nagrade za trenutno stanje agenta. Ukupan broj koraka, odnosno prikupljenih promatranja i poduzetih akcija, koji se moraju napraviti u okolini prije završetka treniranja definiran je parametrom *max_steps*. *Keep_checkpoints* parametar određuje maksimalni broj kontrolnih točaka modela koje se trebaju spremiti.

Također, PPO i SAC algoritmi dijele sljedeće parametre koji pripadaju podskupini hiperparametara: *learning_rate, batch_size, buffer_size, learning_rate_schedule. Learning_rate* parametar opisuje početnu stopu učenja i za oba algoritma ostavljen je na zadanoj vrijednosti 0.0003. Parametar *batch_size* definira broj koji bi trebao biti višestruko manji od parametra *buffer_size* koji određuje broj iskustava koje treba prikupiti prije ažuriranja modela. Za PPO algoritam vrijednosti parametara *batch_size* i *buffer_size* su postavljene na 128 i 2048, a za SAC algoritam su 128 i 50000. Parametar *learning_rate_schedule* određuje kako se stopa učenja mijenja tijekom vremena. Za PPO algoritam postavljena je linearna vrijednost, a za SAC konstantna.

Osim parametara podskupine hiperparametara, PPO i SAC dijele parametre podskupine postavki mreže (engl. *network settings*): *hidden_units, num_layers, normalize, vis_encode_type*. Svi parametri osim parametra *hidden_units* ostavljeni su sa zadanim vrijednostima. *Hidden_units* parametar određuje broj jedinica u skrivenim slojevima neuronske mreže i postavljen je na vrijednost 256. Parametar *num_layers* definira broj skrivenih slojeva u neuronskoj mreži. Zbog rada s diskretnim problemima upravljanja parametar normalizacije imena *normalize* je postavljen na vrijednost *false*. Parametar *vis_encode_type* određuje vrstu kodera za kodiranje vizualnih promatranja.

Parametri nagrađivanja podskupine signala za nagrađivanje (engl. *reward signals*) su zajednički u oba algoritma i ostavljene su im zadane vrijednosti 0.1 za parametar *strength* i 0.99 za *gamma*. Parametar *strength* je faktor kojim se umnožava nagrada koju daje okolina, a *gamma* je faktor popusta za buduće nagrade koje dolaze iz okoline. Opisuje koliko daleko unaprijed agent treba razmišljati o mogućim nagradama.

Osim prethodno navedenih zajedničkih parametara za treniranje agenta, postoje i specifični parametri koji ovise o algoritmu. Prethodno navedeni zajednički parametri i oni specifični, koji će biti u nastavku opisani, prikazani su na slici 5.8. koja prikazuje sadržaj yaml datoteka za PPO i SAC algoritme.

```yaml
behaviors:
  MoveToTarget:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 2048
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 500000
    time_horizon: 64
    summary_freq: 50000
```

```yaml
behaviors:
  MoveToTarget:
    trainer_type: sac
    hyperparameters:
      learning_rate: 0.0003
      batch_size: 128
      buffer_size: 50000
      learning_rate_schedule: constant
      buffer_init_steps: 0
      init_entcoef: 0.1
      save_replay_buffer: false
      tau: 0.005
      steps_per_update: 10.0
      reward_signal_steps_per_update: 10.0
    network_settings:
      hidden_units: 256
      num_layers: 2
      normalize: false
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        strength: 1.0
        gamma: 0.99
    keep_checkpoints: 5
    max_steps: 500000
    time_horizon: 64
    summary_freq: 50000
```

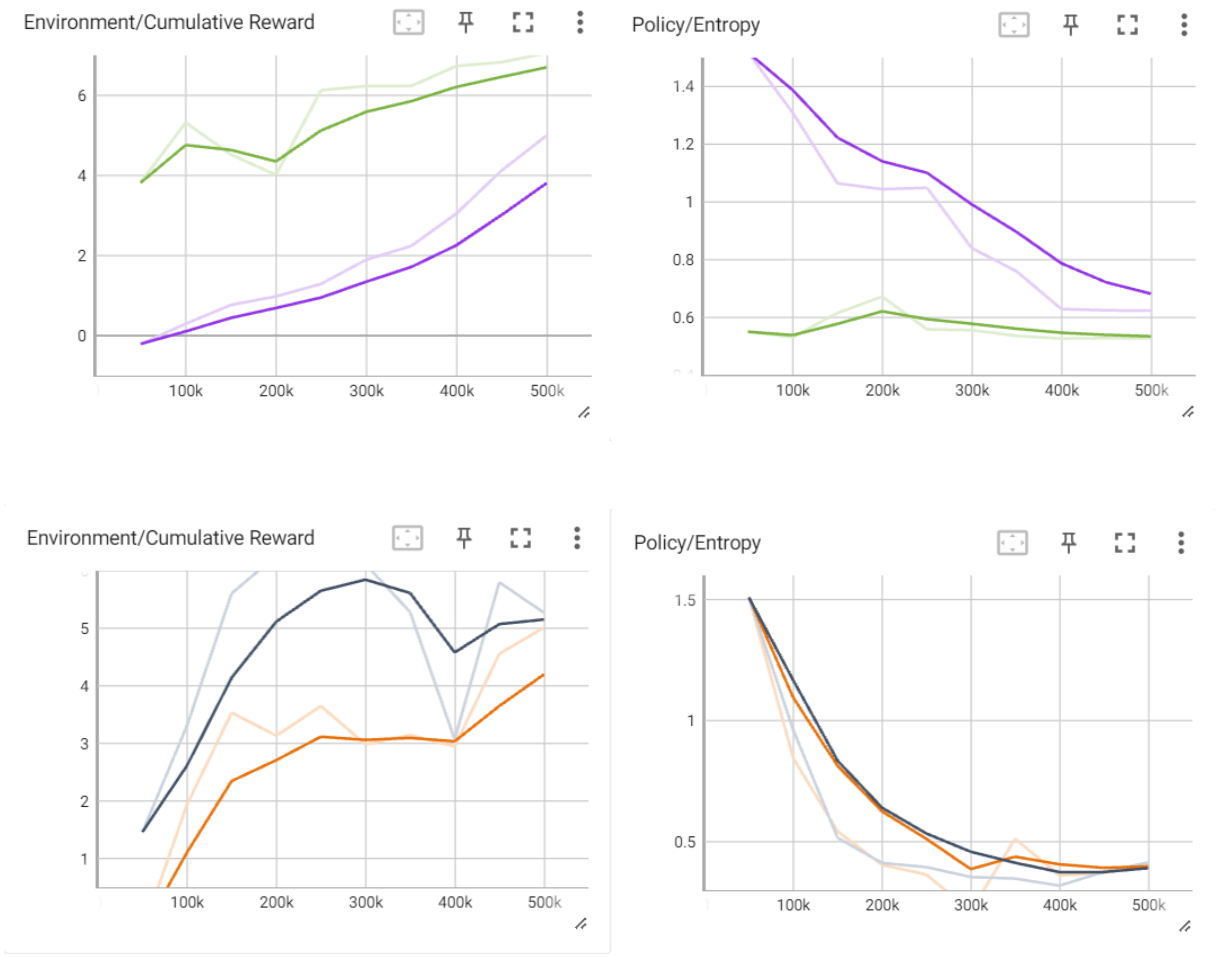
Slika 5.8. Yaml datoteke s hiperparametrima treniranja, lijevo za PPO, desno za SAC

Uz prethodno navedeni zajednički skup parametara za treniranje, specifični parametri koji su korišteni za PPO algoritam su: *beta, epsilon, lambd, num_epoch*. Svi navedeni parametri ostavljeni su kao zadane (engl. *default*) vrijednosti. *Beta* parametar određuje snagu regulacije entropije, koja ponašanja čini slučajnijima. Osigurava da agent pravilno istražuje prostor akcije tijekom treninga. Povećanjem ovog parametra, povećava se i nasumičnost akcija. *Epsilon* parametar utječe na brzinu razvoja algoritma tijekom treniranja. Mala vrijednost ovog parametra rezultira stabilnijim ažuriranjima, ali usporava proces obuke. *Lambd* predstavlja regulacijski parametar koji određuje koliko se agent oslanja na trenutnu procjenu vrijednosti pri izračunu sljedeće procjene vrijednosti. Niske vrijednosti parametra odgovaraju većem oslanjanju na trenutnu procjenu vrijednosti, dok visoke vrijednosti odgovaraju većem oslanjanju na stvarne nagrade ostvarene u okolini. *Num_epoch* parametar definira broj prolaza kroz međuspremnik iskustva prilikom izvođenja optimizacije gradijentnog spusta.

Kao što postoje specifični parametri za PPO algoritam postoje i specifični parametri za SAC algoritam. Specifični parametri koji su korišteni za SAC algoritam su: *buffer_init_steps, init_entcoef, save_replay_buffer, tau, steps_per_update, reward_signal_steps_per_update*. Za parametre *buffer_init_steps, save_replay_buffer* i *tau* postavljene su zadane vrijednosti. *Buffer_init_steps* parametar određuje broj iskustava prikupljenih u međuspremnik prije ažuriranja modela. Parametar *init_entcoef* odgovara početnom entropijskom koeficijentu postavljenom na početku treninga. On opisuje koliko agent treba istraživati na početku treninga. Povećanjem ovog parametra istražuje se više na početku treninga, a smanjenjem dolazi se brže do rješenja. Zbog toga je ovaj parametar postavljen na manju vrijednost 0.1 iz raspona od 0.05 do 0.5. *Save_replay_buffer* je parametar koji određuje hoće li se učitati i spremiti međuspremnik za reprizu iskustva, kao i model pri napuštanju i ponovnom pokretanju treninga. Ovaj parametar je postavljen na *false* jer u suprotnom zauzima veliku količinu memorije. *Tau* parametar određuje koliko često treba ažurirati ciljnu mrežu. U SAC-u postoje dvije neuronske mreže: cilj i politika. Ciljna mreža se koristi za procjenu vrijednosti politike o budućim nagradama u danom stanju i fiksna je dok se mreža politike ažurira. Vrijednosti parametara *steps_per_update* i *reward_signal_steps_per_update* postavljene su na vrijednost 10.0. Parametar *steps_per_update* određuje prosječni omjer poduzetih akcija agenta i ažuriranja politike agenta, a parametar *reward_signal_steps_per_update* određuje broj ažuriranja signala nagrada koji se ažurira svaki put kada je ažurirana politika agenta.

5.3.1. Rezultati treniranja

Treniranje se vrši korištenjem Anaconda Prompt naredbenog retka za pozivanje naredbi. Nakon završetka treniranja, dobivaju se rezultati koji sadrže kopiju konfiguracijske datoteke korištene pri treniranju, model neuronske mreže i TensorBoard prikaz rezultata. Model neuronske mreže koji je dobiven kao rezultat treniranja može se predati u komponentu *Behaviour Parameters* svojstvu model i tako postići da se agent ponaša na temelju naučenog. Na slici 5.10. ljubičastom bojom označeno je prvih 500 tisuća koraka za PPO algoritam, a zelenom drugih 500 tisuća. Za SAC algoritam narančastom bojom označeno je prvih 500 tisuća koraka, a sivom drugih. Iz TensorBoard prikaza može se zaključiti postizanje veće sveukupne nagrade pri korištenju PPO algoritma, uz to što je i njegovo izvođenje bilo trostruko kraće. Iako se entropija u SAC algoritmu puno brže spušta za daljnje treniranje agenta izabran je PPO algoritam.

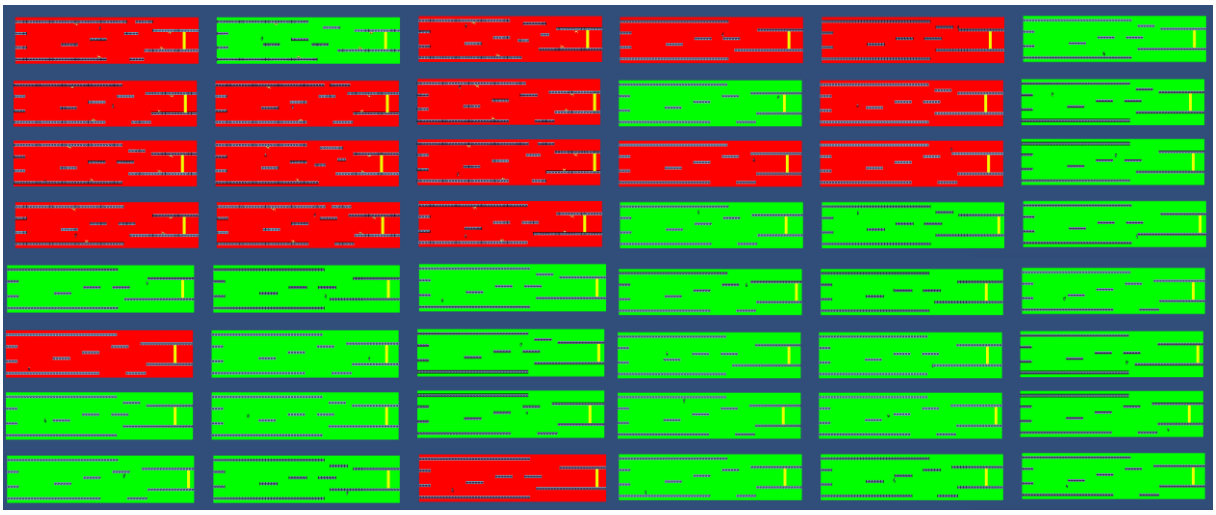


Slika 5.9. TensorBoard prikaz rezultata treniranja

Na slici 5.10. prikazani su rezultati treniranja pomoću PPO algoritma. Izvršeno je pet treniranja po 500 tisuća koraka dok se nije postigao željeni rezultat. Rezultat petog treninga kasnije je korišten za treniranje agenta u naprednoj okolini. Na slici 5.10. se može vidjeti napredak, odnosno povećanje vrijednosti stavke srednje nagrade (engl. *mean reward*) i standardna devijacija nagrade (engl. *standard deviation of reward*). Tijekom treniranja teži se ostvarenju što veće srednje vrijednosti nagrade uz standardnu devijaciju nagrade bližu nuli. Tijek treniranja prikazan je na slici 5.11. gdje se mogu vidjeti rezultati uspješnog i neuspješnog završetka epizode.

```
[INFO] MoveToTarget. Step: 50000. Time Elapsed: 129.022 s. Mean Reward: -0.208. Std of Reward: 1.072. Training.
[INFO] MoveToTarget. Step: 100000. Time Elapsed: 255.087 s. Mean Reward: 0.295. Std of Reward: 1.178. Training.
[INFO] MoveToTarget. Step: 150000. Time Elapsed: 381.371 s. Mean Reward: 0.765. Std of Reward: 0.975. Training.
[INFO] MoveToTarget. Step: 200000. Time Elapsed: 505.478 s. Mean Reward: 0.979. Std of Reward: 0.736. Training.
[INFO] MoveToTarget. Step: 250000. Time Elapsed: 634.892 s. Mean Reward: 1.291. Std of Reward: 0.964. Training.
[INFO] MoveToTarget. Step: 300000. Time Elapsed: 757.826 s. Mean Reward: 1.894. Std of Reward: 0.814. Training.
[INFO] MoveToTarget. Step: 350000. Time Elapsed: 884.013 s. Mean Reward: 2.239. Std of Reward: 1.107. Training.
[INFO] MoveToTarget. Step: 400000. Time Elapsed: 1009.273 s. Mean Reward: 3.047. Std of Reward: 1.876. Training.
[INFO] MoveToTarget. Step: 450000. Time Elapsed: 1133.537 s. Mean Reward: 4.120. Std of Reward: 2.494. Training.
[INFO] MoveToTarget. Step: 500000. Time Elapsed: 1260.128 s. Mean Reward: 4.996. Std of Reward: 2.684. Training.
[INFO] MoveToTarget. Step: 50000. Time Elapsed: 205.755 s. Mean Reward: 5.818. Std of Reward: 2.318. Training.
[INFO] MoveToTarget. Step: 100000. Time Elapsed: 398.041 s. Mean Reward: 6.004. Std of Reward: 1.956. Training.
[INFO] MoveToTarget. Step: 150000. Time Elapsed: 588.664 s. Mean Reward: 6.553. Std of Reward: 1.998. Training.
[INFO] MoveToTarget. Step: 200000. Time Elapsed: 780.430 s. Mean Reward: 6.846. Std of Reward: 1.926. Training.
[INFO] MoveToTarget. Step: 250000. Time Elapsed: 970.058 s. Mean Reward: 6.678. Std of Reward: 2.069. Training.
[INFO] MoveToTarget. Step: 300000. Time Elapsed: 1159.817 s. Mean Reward: 5.902. Std of Reward: 2.124. Training.
[INFO] MoveToTarget. Step: 350000. Time Elapsed: 1350.241 s. Mean Reward: 6.798. Std of Reward: 1.738. Training.
[INFO] MoveToTarget. Step: 400000. Time Elapsed: 1540.321 s. Mean Reward: 6.857. Std of Reward: 1.654. Training.
[INFO] MoveToTarget. Step: 450000. Time Elapsed: 1730.937 s. Mean Reward: 7.327. Std of Reward: 0.743. Training.
[INFO] MoveToTarget. Step: 500000. Time Elapsed: 1923.792 s. Mean Reward: 7.229. Std of Reward: 0.899. Training.
[INFO] MoveToTarget. Step: 50000. Time Elapsed: 205.755 s. Mean Reward: 5.818. Std of Reward: 2.318. Training.
[INFO] MoveToTarget. Step: 100000. Time Elapsed: 398.041 s. Mean Reward: 6.004. Std of Reward: 1.956. Training.
[INFO] MoveToTarget. Step: 150000. Time Elapsed: 588.664 s. Mean Reward: 6.553. Std of Reward: 1.998. Training.
[INFO] MoveToTarget. Step: 200000. Time Elapsed: 780.430 s. Mean Reward: 6.846. Std of Reward: 1.926. Training.
[INFO] MoveToTarget. Step: 250000. Time Elapsed: 970.058 s. Mean Reward: 6.678. Std of Reward: 2.069. Training.
[INFO] MoveToTarget. Step: 300000. Time Elapsed: 1159.817 s. Mean Reward: 5.902. Std of Reward: 2.124. Training.
[INFO] MoveToTarget. Step: 350000. Time Elapsed: 1350.241 s. Mean Reward: 6.798. Std of Reward: 1.738. Training.
[INFO] MoveToTarget. Step: 400000. Time Elapsed: 1540.321 s. Mean Reward: 6.857. Std of Reward: 1.654. Training.
[INFO] MoveToTarget. Step: 450000. Time Elapsed: 1730.937 s. Mean Reward: 7.327. Std of Reward: 0.743. Training.
[INFO] MoveToTarget. Step: 500000. Time Elapsed: 1923.792 s. Mean Reward: 7.229. Std of Reward: 0.899. Training.
[INFO] MoveToTarget. Step: 50000. Time Elapsed: 202.197 s. Mean Reward: 6.463. Std of Reward: 1.616. Training.
[INFO] MoveToTarget. Step: 100000. Time Elapsed: 392.891 s. Mean Reward: 6.812. Std of Reward: 1.498. Training.
[INFO] MoveToTarget. Step: 150000. Time Elapsed: 584.695 s. Mean Reward: 7.205. Std of Reward: 1.435. Training.
[INFO] MoveToTarget. Step: 200000. Time Elapsed: 774.592 s. Mean Reward: 6.979. Std of Reward: 1.650. Training.
[INFO] MoveToTarget. Step: 250000. Time Elapsed: 960.656 s. Mean Reward: 6.549. Std of Reward: 1.441. Training.
[INFO] MoveToTarget. Step: 300000. Time Elapsed: 1146.213 s. Mean Reward: 6.998. Std of Reward: 1.200. Training.
[INFO] MoveToTarget. Step: 350000. Time Elapsed: 1334.483 s. Mean Reward: 7.261. Std of Reward: 1.091. Training.
[INFO] MoveToTarget. Step: 400000. Time Elapsed: 1519.468 s. Mean Reward: 7.409. Std of Reward: 0.747. Training.
[INFO] MoveToTarget. Step: 450000. Time Elapsed: 1704.146 s. Mean Reward: 7.485. Std of Reward: 0.727. Training.
[INFO] MoveToTarget. Step: 500000. Time Elapsed: 1892.249 s. Mean Reward: 7.358. Std of Reward: 1.008. Training.
[INFO] MoveToTarget. Step: 50000. Time Elapsed: 198.710 s. Mean Reward: 7.062. Std of Reward: 1.273. Training.
[INFO] MoveToTarget. Step: 100000. Time Elapsed: 386.780 s. Mean Reward: 5.830. Std of Reward: 1.895. Training.
[INFO] MoveToTarget. Step: 150000. Time Elapsed: 577.621 s. Mean Reward: 7.133. Std of Reward: 1.351. Training.
[INFO] MoveToTarget. Step: 200000. Time Elapsed: 767.071 s. Mean Reward: 7.174. Std of Reward: 1.392. Training.
[INFO] MoveToTarget. Step: 250000. Time Elapsed: 958.401 s. Mean Reward: 7.177. Std of Reward: 1.018. Training.
[INFO] MoveToTarget. Step: 300000. Time Elapsed: 1152.311 s. Mean Reward: 7.490. Std of Reward: 0.713. Training.
[INFO] MoveToTarget. Step: 350000. Time Elapsed: 1345.600 s. Mean Reward: 7.330. Std of Reward: 1.049. Training.
[INFO] MoveToTarget. Step: 400000. Time Elapsed: 1536.612 s. Mean Reward: 7.413. Std of Reward: 0.917. Training.
[INFO] MoveToTarget. Step: 450000. Time Elapsed: 1730.605 s. Mean Reward: 7.229. Std of Reward: 1.313. Training.
[INFO] MoveToTarget. Step: 500000. Time Elapsed: 1960.317 s. Mean Reward: 7.406. Std of Reward: 0.669. Training.
```

Slika 5.10. Podaci o treniranju iz Anaconda Prompta



Slika 5.11. Prikaz jednostavne okoline tijekom treniranja

5.4. Izrada napredne okoline za treniranje agenta

Između jednostavne okoline i napredne okoline nema mnogo razlika. Jedna od razlika je način generiranja drugog srednjeg odjeljka na početku epizode. To je definirano u funkciji *ChooseMiddleSection* koja se poziva na objektu klase *MiddleSectionManager* na početku epizode (Programski kod 5.13.). Napredna okolina ima listu odjeljaka veličine 15, od kojih se bira samo 12. Potreban je veliki broj različitih kombinacija okoline kako bi treniranje agenta bilo uspješno. Mogućnost različitih kombinacija okoline računa se pomoću formule (5-1) za varijacije bez ponavljanja:

$$V_n^k = \frac{n!}{(n-k)!} = \frac{15!}{(15-12)!} = 2.1794 \cdot 10^{11} \quad (5\text{-}1)$$

gdje je *V* varijacija, malo *n* broj članova skupa, a malo *k* broj elementa koji se bira iz skupa *n*. Uvrstivši brojeve $n = 15$ i $k = 12$ dobije se broj mogućih kombinacija $2.1794 \cdot 10^{11}$ što je zadovoljavajuće za naprednu okolinu.

Linija ***Kod***

```
1:      public void ChooseMiddleSection()
2:      {
3:          for (int i = 0; i < middleSections.Count; i++)
4:          {
5:              middleSections[i].SetActive(false);
6:          }
7:          if (middleSections.Count == 15)
8:          {
9:              Shuffle(middleSections);
10:             for (int i = 0; i < 12; i++)
11:             {
                   middleSections[i].transform.localPosition = new Vector3(i
12:     * 16, 0);
13:                 middleSections[i].SetActive(true);
14:             }
15:         }
16:         else
17:         {
                chosenMiddleSection = Random.Range(0, middleSections.Count);
18:             middleSections[chosenMiddleSection].SetActive(true);
19:         }
20:     }
```

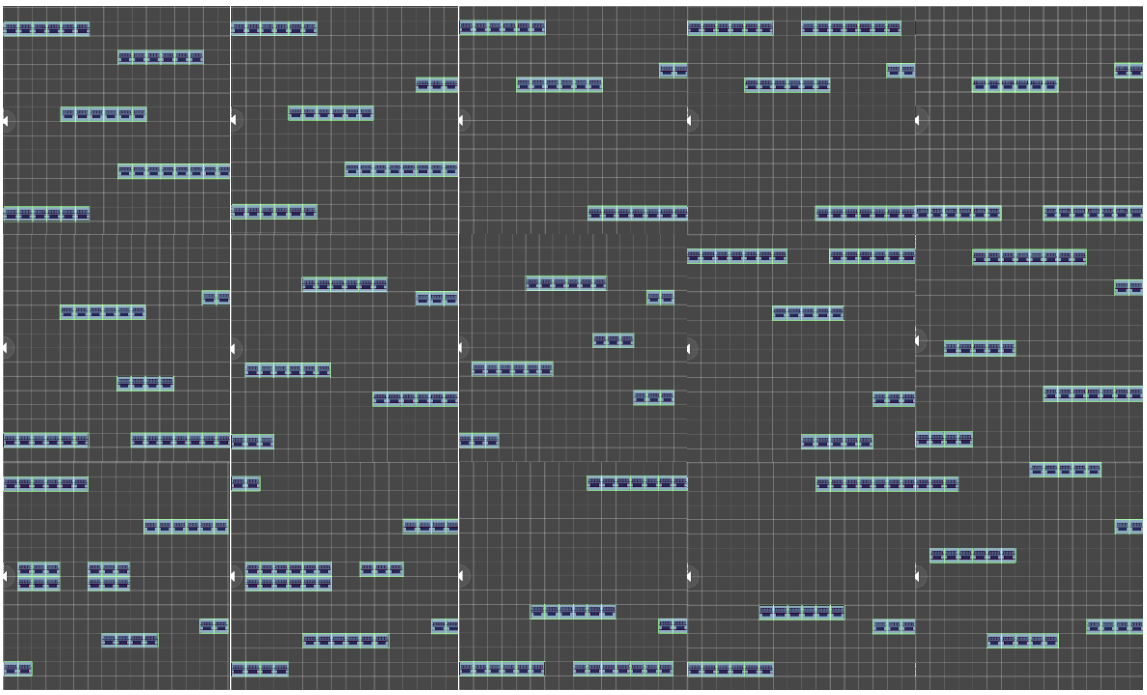
Programski kod 5.13. Isječak funkcija ChooseMiddleSection iz klase MiddleSectionManager

Nasumičnost objekata liste postiže se korištenjem Fisher-Yates algoritma miješanja [33]. Ovaj algoritam ima složenost *O(n)*, gdje je *n* broj elemenata koji se miješaju. Fisher-Yates algoritam miješanja uzima listu svih elemenata niza i kontinuirano, od zadnjeg elementa prema prvom, nasumično izvlači elemente i mijenja njihova mjesta u listi. Ovaj algoritam proizvodi nepristranu permutaciju elemenata jer je svaka jednako vjerojatna. Opisana funkcija *Shuffle* (Programski kod 5.14.) koristi listu različitih objekata odjeljaka koji su prikazani na slici 5.12.

Linija ***Kod***

```
        private List<GameObject> Shuffle(List<GameObject>
1:      middleSectionsToShuffle)
2:      {
3:          for (int i = middleSectionsToShuffle.Count - 1; i > 0; i--)
4:          {
5:              int randNum = random.Next(i + 1);
6:              GameObject temp = middleSectionsToShuffle[randNum];
                middleSectionsToShuffle[randNum] =
7:      middleSectionsToShuffle[i];
8:              middleSectionsToShuffle[i] = temp;
9:          }
10:         return middleSectionsToShuffle;
11:     }
```

Programski kod 5.14. Isječak funkcija Shuffle iz klase MiddleSectionManager



Slika 5.12. Prikaz svih odjeljaka drugog srednjeg odjeljka

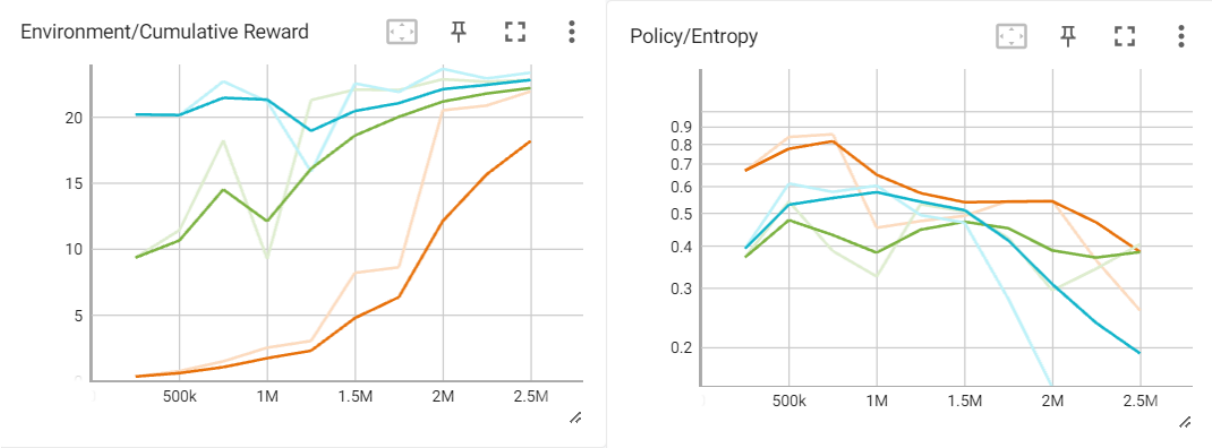
Za treniranje agenta u naprednoj okolini varijabla maksimalnog broja koraka (engl. *max step*) koje agent ostvaruje na skripti *GravityAgent* je promijenjena u 25000. Zbog toga su vrijednosti hiperparametara u yaml datoteci za PPO algoritam promijenjeni. Vrijednost parametra *max_steps* postavljena je na 2500000, a vrijednost *summary_freq* na 250000 (Slika 5.13.).

```yaml
behaviors:
  MoveToTarget:
    trainer_type: ppo
    hyperparameters:
      learning_rate: 0.0003
      batch_size: 128
      buffer_size: 2048
      learning_rate_schedule: linear
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
    network_settings:
      hidden_units: 256
      num_layers: 2
      normalize: false
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        strength: 1.0
        gamma: 0.99
    keep_checkpoints: 5
    max_steps: 2500000
    time_horizon: 64
    summary_freq: 250000
```

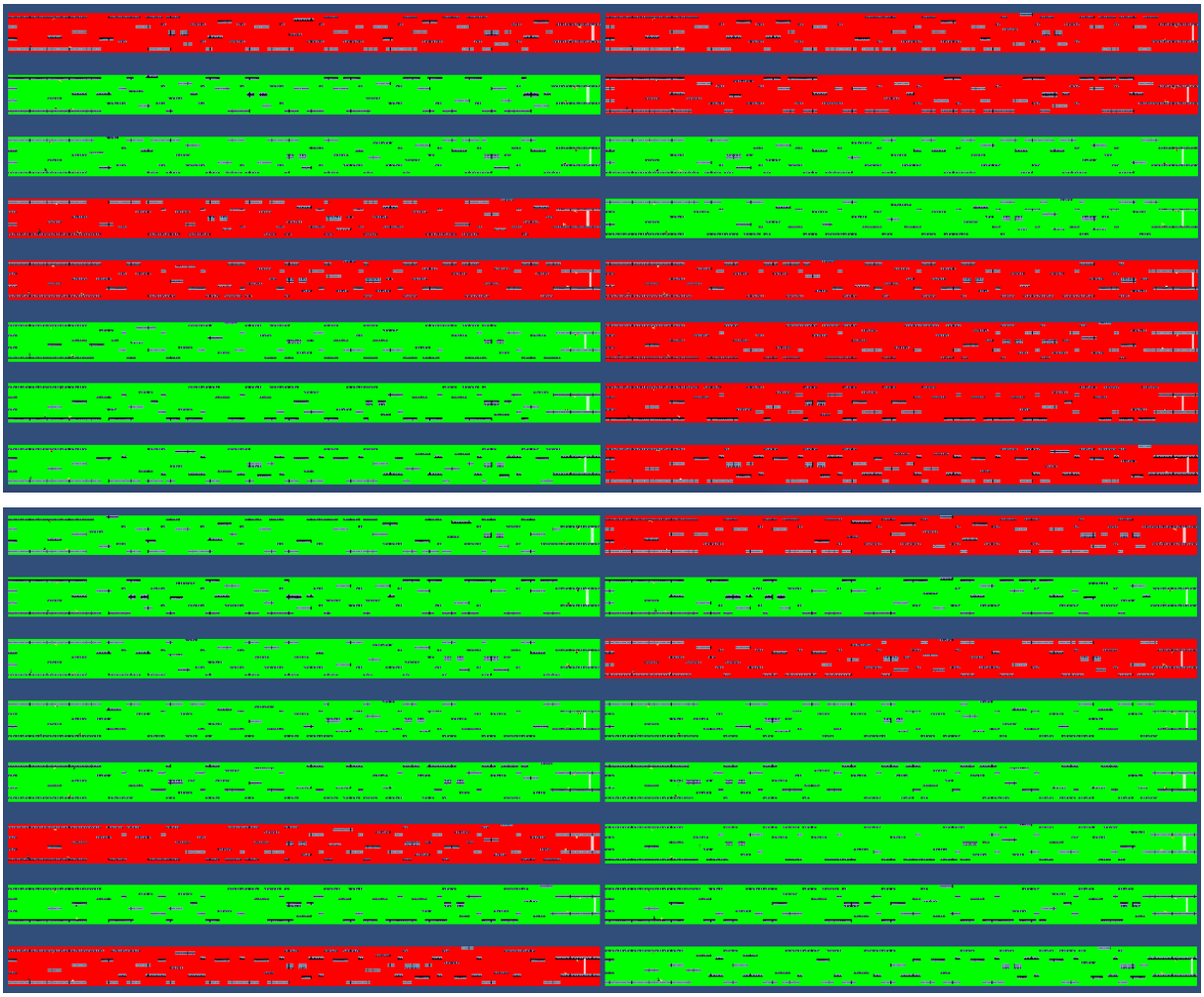
Slika 5.13. Yaml datoteke s hiperparametrima treniranja za PPO algoritam

5.4.1. Rezultati treniranja

Na temelju krajnjeg treniranja u jednostavnoj okolini, nastavlja se treniranje agenta u naprednoj okolini. Na slici 5.14. narančastom bojom prikazano je prvih 2500 tisuća koraka, zelenom drugih, a plavom trećih. Tijekom trećeg treniranja postiže se zadovoljavajuća sveukupna vrijednost nagrade i time se završava treniranje agenta. Iz TensorBoard prikaza može se zaključiti da je entropija treniranja nestabilna, a to se događa zbog izgleda odjeljaka. Kako bi se odjeljci mogli nasumično nastavljati jedan na drugi na određenim položajima se trebaju nalaziti platforme. Platforme na krajevima zadnjeg odjeljaka koje idu predaleko od centralne linije rezultiraju negativnim rezultatom, ali to ne utječe na rezultate pri procjeni istreniranog agenta jer su te okoline napravljene od drugačijeg srednjeg odjeljka. Na slici 5.15. prikazane su napredne okoline tijekom treniranja agenta.



Slika 5.14. TensorBoard prikaz rezultata treniranja u naprednoj okolini



Slika 5.15. *Prikaz napredne okoline tijekom treniranja*

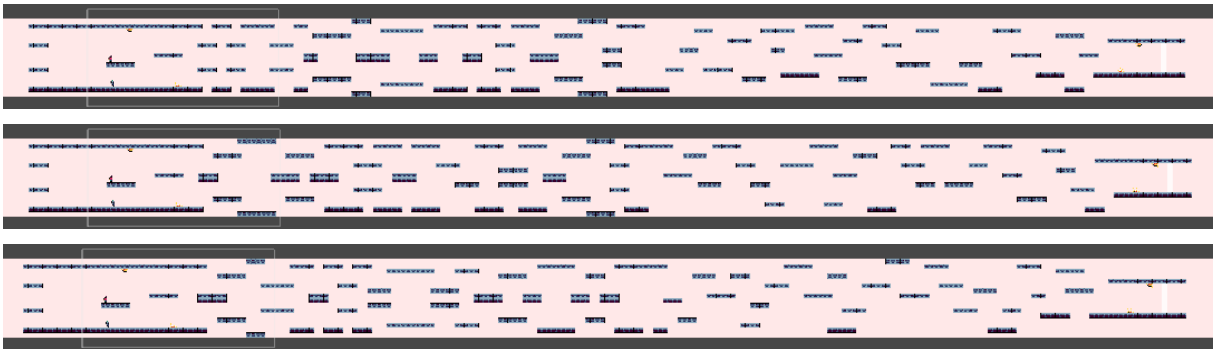
U tablici 5.1. prikazani su rezultati treniranja pomoću PPO algoritma u naprednoj okolini. Kako bi se postigli željeni rezultati izvršeno je tri treniranja po 2500 tisuća koraka. Proučavajući stupac stavke srednje nagrade (engl. *mean reward*) može se vidjeti napredak agenta tijekom treniranja. Rezultat prvog treniranja nije imao zadovoljavajuću srednju nagradu, a rezultat drugog treniranja je imao u prosjeku preveliku standardnu devijaciju nagrade. Zbog tih razloga, za rezultat treniranja koristi se model neuronske mreže *MoveToTarget-2499971* iz trećeg treniranja agenta.

Tablica 5.1. Rezultati treniranja u naprednoj okolini

Identifikator treninga/Rezultati	**adv_ppo_01b _movetotarget**	**adv_ppo_01g _movetotarget**	**adv_ppo_02i _movetottarget**
Srednja nagrada			
Koraci:			
250000	0.3564	9.366	20.211
500000	0.7726	11.446	20.147
750000	1.505	18.244	22.721
1000000	2.545	9.285	21.185
1250000	3.057	21.302	15.871
1500000	8.215	22.095	22.564
1750000	8.633	22.064	21.918
2000000	20.527	22.879	23.671
2250000	20.888	22.693	22.940
2500000	21.955	22.833	23.375
Standardna devijacija nagrade			
Koraci:			
250000	1.178	8.665	6.396
500000	0.975	8.606	7.325
750000	0.914	7.811	4.695
1000000	1.107	8.005	5.398
1250000	1.876	5.236	8.731
1500000	9.636	3.219	2.648
1750000	9.680	3.750	4.327
2000000	7.029	2.807	2.261
2250000	6.617	3.120	2.210
2500000	6.255	1.770	1.776

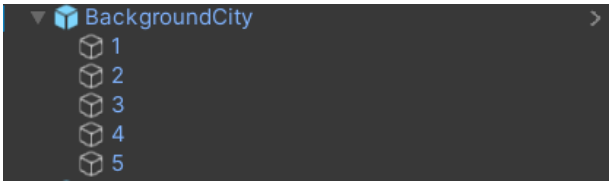
5.5. Izrada okoline za procjenu istreniranog agenta

Za procjenu istreniranog agenta izrađene su tri razine čiji srednji odjeljci nisu međusobno jednaki, a nisu ni jednaki odjeljcima korištenim tijekom treniranja agenta. Provedeno je testiranje pojedine razine postavljajući model neuronske mreže *MoveToTarget-2499971*. Istrenirani agent uspješno prolazi navedene razine, na slici 5.16. prikazane su sve tri razine.



Slika 5.16. *Prikaz napredne okoline tijekom treniranja*

Kako bi se lakše moglo razlikovati razine, kreirane su pozadine s paralaksa efektom (engl. *parallax effect*). Paralaksa predstavlja prividnu promjenu položaja promatranog objekta s promjenom mjesta promatrača, gdje se sve udaljeniji objekti sporije gibaju u suprotnom smjeru. U scenu je dodan *GameObject* naziva *BackgroundCity* čijim je dijete objektima pridružena kao komponenta skripta *Parallax* (Slika 5.17.).



Slika 5.17. *Prikaz hijerarhija objekata pozadine*

Klasi *Parallax* predaju se objekti kamere i agenta koji se koriste za računanje faktora paralaksa. Kamera sadrži komponentu skripte *TimeScalar*, koja tri puta ubrzava vrijeme razina, i komponentu skripte *CameraManager*, koja joj omogućuje slijeđenje igrača kroz razinu. Varijabla dvodimenzionalnog vektora putovanja (engl. *travel*) pri svakom korištenju računa se kao razlika položaja kamere i dvodimenzionalnog vektora početnog položaja objekta pozadine na početku razine. Za računanje faktora paralaksa potrebno je izračunati apsolutnu vrijednost omjera varijabli udaljenosti od agenta i rezne ravnine (engl. *clipping plane*) kamere. Udaljenost od agenta računa se kao razlika položaja objekta pozadine u z ravnini i položaja agenta. Rezna ravnina se računa

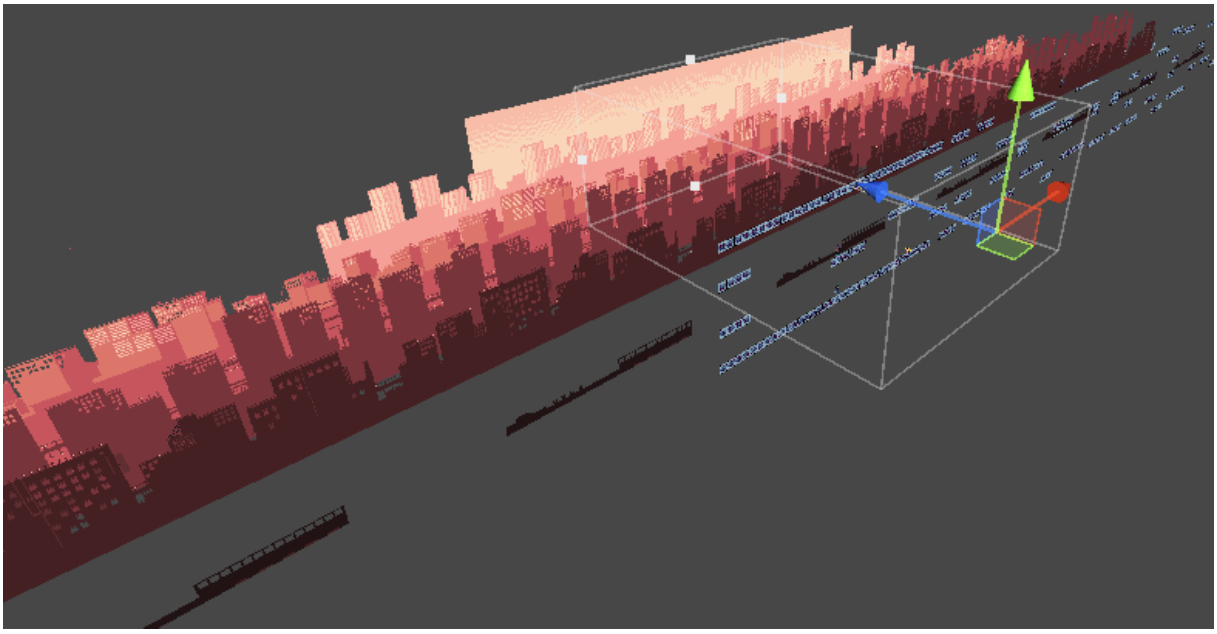
kao zbroj položaja kamere u z ravnini i položaja udaljenije rezne ravnine kamere koja ima vrijednost 50. Pri pomicanju igrača u razini, računa se novi položaj objekta pozadine, kao zbroj njegovog početnog položaja i umnoška varijable putovanja i faktora paralaksa (Programski kod 5.15.).

Linija ***Kod***

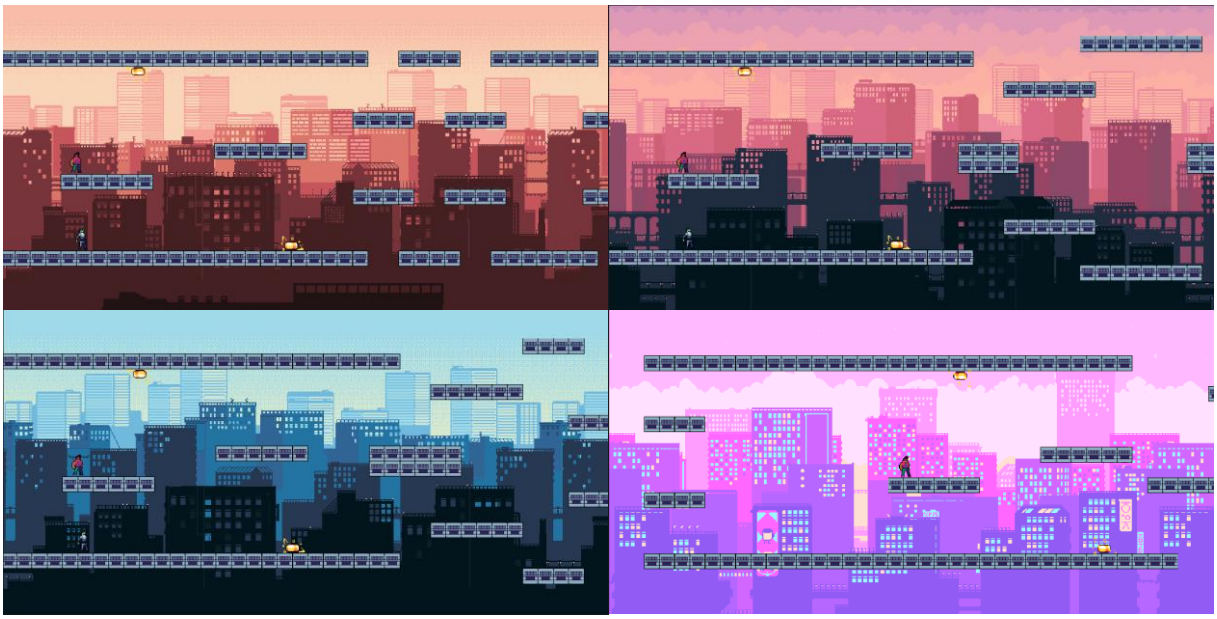
```
1:      using UnityEngine;
2:
3:      public class Parallax : MonoBehaviour
4:      {
5:          [SerializeField] private Camera cam;
6:          [SerializeField] private Transform agent;
7:          private Vector2 startPosition;
8:          private float startZ;
            private Vector2 travel => (Vector2)cam.transform.position -
9:      startPosition;
            private float distanceFromAgent => transform.position.z -
10:     agent.position.z;
            private float clippingPlane => cam.transform.position.z +
11:     cam.farClipPlane;
            private float parallaxFactor => Mathf.Abs(distanceFromAgent /
12:     clippingPlane);
13:
14:         void Start()
15:         {
16:             startPosition = transform.position;
17:             startZ = transform.position.z;
18:         }
19:         void Update()
20:         {
                Vector2 newPosition = startPosition + travel *
21:     parallaxFactor;
                transform.position = new Vector3(newPosition.x,
22:     newPosition.y, startZ);
23:         }
24:     }
```

Programski kod 5.15. Klasa Parallax

Postavljanje u trodimenzionalnom prostoru objekata pozadine vidljivo je na slici 5.18. Objekti pozadine postavljeni su na 95%, 90%, 75%, 65%, 50% i 25% udaljenosti od agenta ako ima šest objekata. U slučaju korištenja pet objekata pozadine, ne postavlja se objekt na 65% udaljenosti od agenta. Udaljenost od agenta do udaljenije rezne ravnine je 40 prostornih jedinica, pa su prethodno definirane udaljenosti jednake 38, 36, 30, 26, 20 i 10. Na slici 5.19. prikazan je dvodimenzionalni izgled prve tri razine i razina za treniranje nakon postavljanja objekata pozadine.



Slika 5.18. Prikaz trodimenzionalnog položaja objekata pozadine



Slika 5.19. Dvodimenzionalni prikaz razina, redom prve, druge i treće razine, te dolje desno razine za treniranje

Na slici 5.19. prikazan je i objekt igrača koji je pridružen svim razinama. Ovom objektu pridružena je skripta *GravityPlayer* koja kao skripta *GravityAgent* nasljeđuje sučelja *IScenes* i *IGravity*. Najveća razlika u skriptama je funkcija *OnTriggerEnter2D*. Ona se ne koristi za određivanje kraja epizode već poziva funkciju *ResetAgent* na objektu klase *GravityAgent* koja završava epizodu. Zbog toga što se agent ne koristi u razini za treniranje igrača, prije poziva funkcije za završavanje epizode ispituje se da agent nije *null* vrijednost.

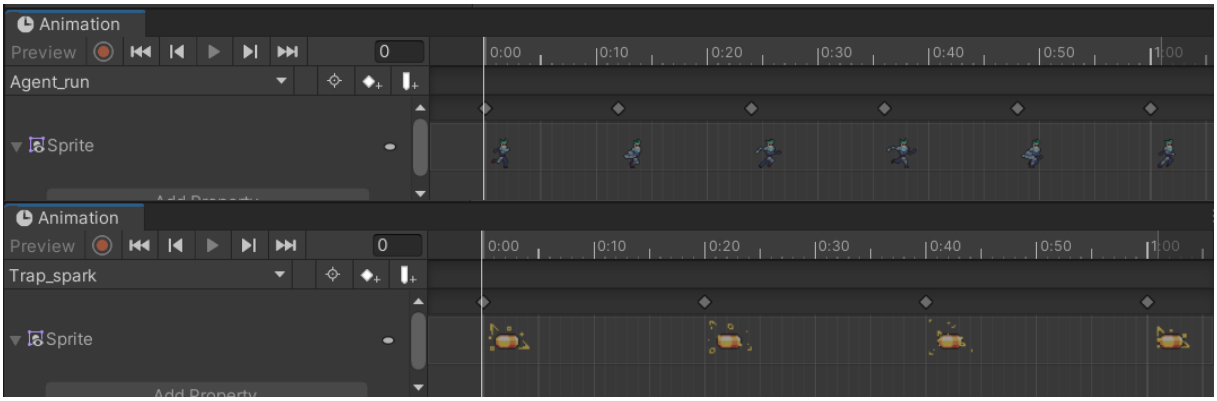
U slučaju sudara s objektom provjerava se njegova oznaka i ako je jednaka *Target,* učitava se sljedeća razina. Ako je trenutna razina zadnja ili trening razina, onda se učitava scena glavnog izbornika (engl. *main menu*). Pri detekciji sudara s objektima oznaka *Border* ili *Trap* učitava se ponovno aktivna razina (Programski kod 5.16.).

Linija Kod

```
1:      private void OnTriggerEnter2D(Collider2D collider)
2:      {
3:          if (collider.gameObject.CompareTag("Target"))
4:              switch (SceneManager.GetActiveScene().name)
5:              {
6:                  case "Level_1":
7:                      SceneManager.LoadScene("Level_2");
8:                      break;
9:                  case "Level_2":
10:                     SceneManager.LoadScene("Level_3");
11:                     break;
12:                 default:
13:                     SceneManager.LoadScene("MainMenu");
14:                     break;
15:             };
        if (collider.gameObject.CompareTag("Border") ||
16:     collider.gameObject.CompareTag("Trap"))
17:         {
18:             SceneManager.LoadScene(SceneManager.GetActiveScene().name);
19:         }
20:         ResetAgent();
21:     }
```

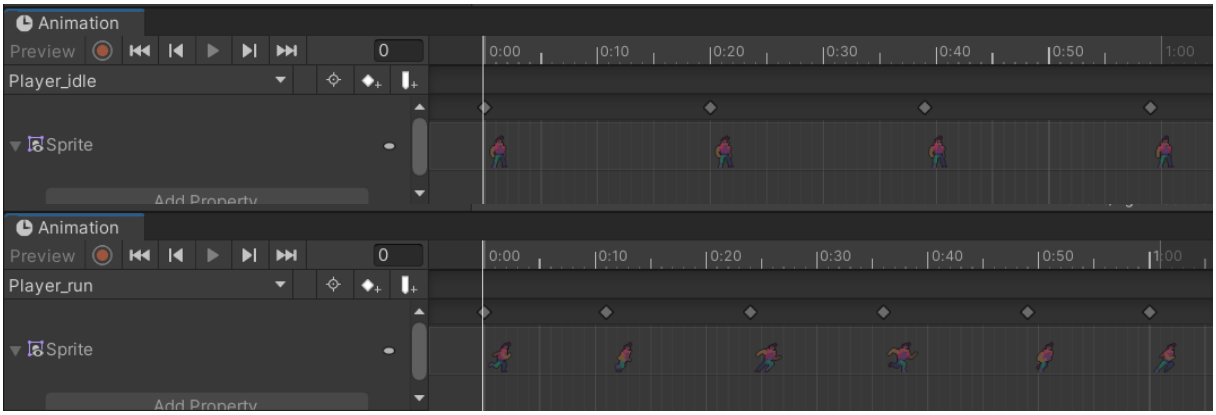
Programski kod 5.16. Isječak funkcije OnTriggerEnter2D iz klase GravityPlayer

Nakon ispitivanja agenta, napravljene su animacije za agenta, zamke i igrača. Za agenta i zamku napravljene su animacije zadanog stanja, gdje je to animacija trčanja za agenta i animacija iskrenja za zamku. Slike korištene za animaciju zamke uređene su u GIMP uređivaču slika. Promijenjena im je boja iz plave u žutu kako bi bile bolje vidljive u razinama. Za oba objekta u upravljaču animatora (engl. *animator controller*) svojstva su ostavljena na zadana. Na slici 5.20. prikazane su animacije redom za agenta, pa za zamku. Obje animacije traju jednu sekundu. Animacija trčanja agenta sadrži šest slika što ju čini bržom od animacije iskrenja zamke koja ima četiri slike.

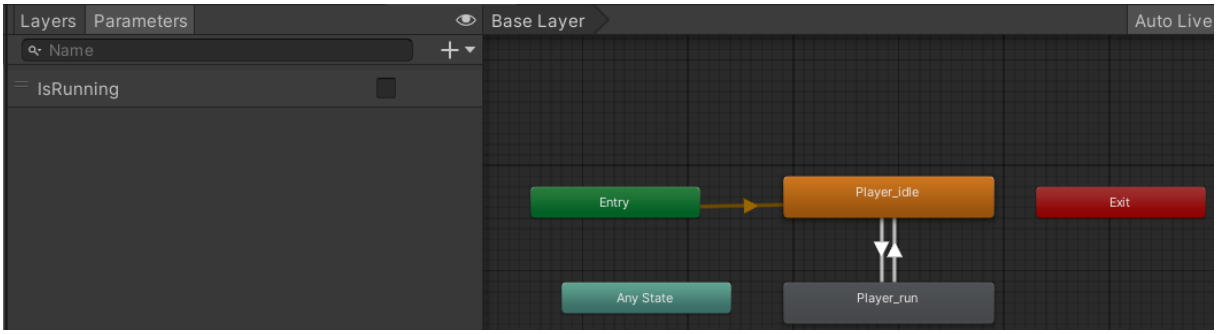


Slika 5.20. Animacija trčanja za agenta i animacija iskrenja za zamku

Za igrača osim animacije trčanja, izrađena je animacija praznog hoda kada se igrač ne pomiče u određenom smjeru već stoji na mjestu. Na slici 5.21. prikazane su animacije igrača. Kako bi se upravljalo izmjenama tih animacija stvoren je parametar *bool* tipa naziva *IsRunning*. Ova varijabla nalazi se u upravljaču animatora. Kao zadana animacija postavljena je animacija praznog hoda, a iz nje se prelazi u animaciju trčanja ako je varijabla *IsRunning* jednaka istini. Za prelazak iz animacije trčanja u animaciju praznog hoda varijabla *IsRunning* treba biti jednaka neistini. Ovi prijelazi prikazani su na slici 5.22.



Slika 5.21.Animacija trčanja i praznog hoda za igrača



Slika 5.22. Animacija trčanja za agenta i animacija iskrenja za zamku

Promjene varijable *IsRunning* vrše se u funkciji *Update* skripte *GravityPlayer*. Igraču je omogućeno pritiskom tipke *escape* vraćanje u glavni izbornik. Time se završava epizoda. Vrijednost varijable *IsRunning* postavlja se na neistinu sve dok se ne pritisnu tipke za pomicanje lijevo-desno, zatim se njena vrijednost mijenja u istinu.

Linija Kod

```
1:      void Update()
2:      {
3:          if (Input.GetKeyDown(KeyCode.Escape))
4:          {
5:              ResetAgent();
6:              SceneManager.LoadScene(IScenes.Name.MainMenu.ToString());
7:          }
8:          animator.SetBool("IsRunning", false);
9:          if (IsGrounded())
10:         {
11:             if (Input.GetKey(KeyCode.A))
12:         {
13:                 if (facingRight)
14:                 {
15:                     Flip();
16:                 }
                    transform.localPosition += Vector3.left * speed *
17:     Time.deltaTime;
18:                 animator.SetBool("IsRunning", true);
19:             }
20:             else if (Input.GetKey(KeyCode.D))
21:             {
22:                 if (!facingRight)
23:                 {
24:                     Flip();
25:                 }
                    transform.localPosition += Vector3.right * speed *
26:     Time.deltaTime;
27:                 animator.SetBool("IsRunning", true);
28:             }
29:             else if (Input.GetKeyDown(KeyCode.W))
30:             {
31:                 GravitySwitch();
32:             }
33:         }
34:     }
```

Programski kod 5.17. Isječak funkcije Update iz klase GravityPlayer

5.6. Izrada glavnog izbornika

Pozadina glavnog izbornika predstavlja objekt čiji se dijelovi pomiču različitim brzinama. Što je objekt bliže u perspektivi korisnika, on ima veću brzinu. Promjena brzine objekta pozadine određena je dodatkom komponente skripte *BackgroundScroller*. Za postizanje pomicanja pozadina korištena je funkcija *SetTextureOffset* na objektu materijala. Ovoj funkciji je predana vrijednost naziva teksture i dvodimenzionalni vektor pomaka. Vektor pomaka predstavlja sumu prethodnih pomaka s umnoškom proteklog vremena od zadnjeg vremenskog okvira (engl. *frame*) u sekundama i postavljene brzine koji je dijeljen s deset.

Linija ***Kod***

```
1:      public class BackgroundScroller : MonoBehaviour
2:      {
3:          [Range(0f, 1f)]
4:          public float speed = 1f;
5:          private float offset;
6:          private Material material;
7:          void Start()
8:          {
9:              material = GetComponent<Renderer>().material;
10:         }
11:         void Update()
12:         {
13:             offset += (Time.deltaTime * speed) / 10f;
                material.SetTextureOffset("_MainTex", new Vector2(offset,
14:     0));
15:         }
16:     }
```

Programski kod 5.18. Klasa BackgroundScroller

Scena glavnog izbornika sadrži objekt *Canvas* koji se sastoji od objekata glavnog izbornika i objekata opcija. Vidljivost ovih objekata se izmjenjuje. Hijerarhija objekta *Canvas* prikazana je na slici 5.23. Objekt glavnog izbornika sadrži objekt teksta naslova igre i četiri objekta tipki, a objekt opcija sadrži objekt ploče koji obuhvaća dva objekta teksta, za opcije i glasnoću, tri objekta slikovnih tipki, jedan objekt klizača i jedan objekt tipke.

Slika 5.23. Animacija trčanja za agenta i animacija iskrenja za zamku

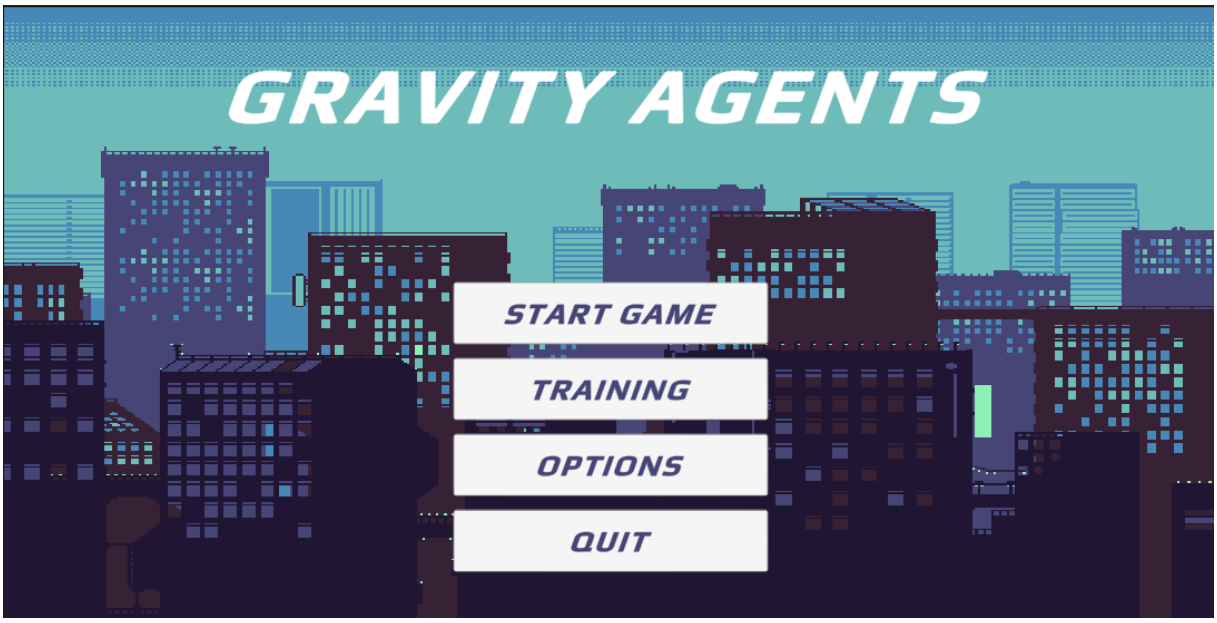
Objektu glavnog izbornika pridružena je kao komponenta skripta *MainMenu*. Klasa *MainMenu* nasljeđuje sučelja *IScenes* i *IQuit*. Sučelje *IQuit* definira metodu *QuitGame* koju ona treba implementirati. U ovoj klasi opisane su funkcije koje se pokreću na pritisak tipki. Funkcija *StartGame* učitava prvu razinu, a funkcija *StartTraining* učitava razinu za treniranje (Programski kod 5.19.).

Linija ***Kod***

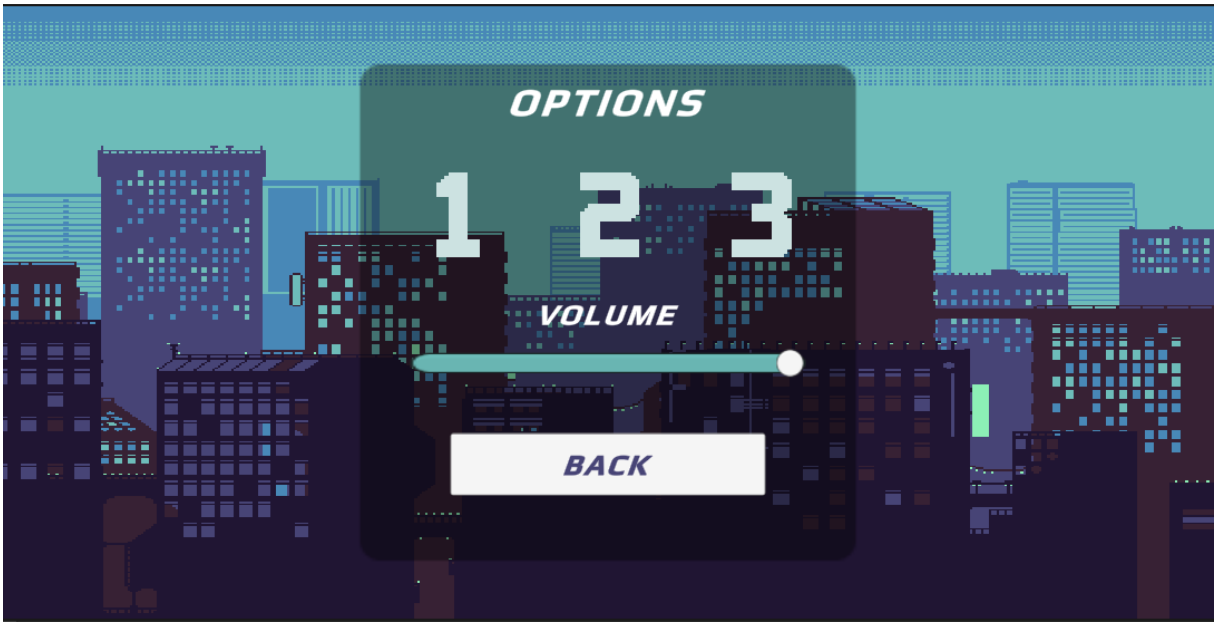
```
1:      using UnityEngine;
2:      using UnityEngine.SceneManagement;
3:
4:      public class MainMenu : MonoBehaviour, IScenes, IQuit
5:      {
6:          public void StartGame()
7:          {
8:              SceneManager.LoadScene(IScenes.Name.Level_1.ToString());
9:          }
10:         public void StartTraining()
11:         {
12:           SceneManager.LoadScene(IScenes.Name.TrainingLevel.ToString());
13:         }

14:         public void QuitGame()
15:         {
16:             Debug.Log("QUIT!");
17:             Application.Quit();
18:         }
19:     }
```

Programski kod 5.19. Klasa MainMenu

Izgled glavnog izbornika prikazan je na slici 5.24. Objektu tipke započni igru (engl. *start game*) na klik predaje se funkcija *StartGame*, objektu tipke trening (engl. *training*) predaje se funkcija *StartTraining*, a objektu tipke odustani (engl. *quit*) funkcija *QuitGame*. Na klik objekta tipke opcije (engl. *options*), postavlja se objekt *Options* kao aktivan, a deaktivira se objekt *MainMenu*. Slika 5.25. prikazuje izgled opcija.

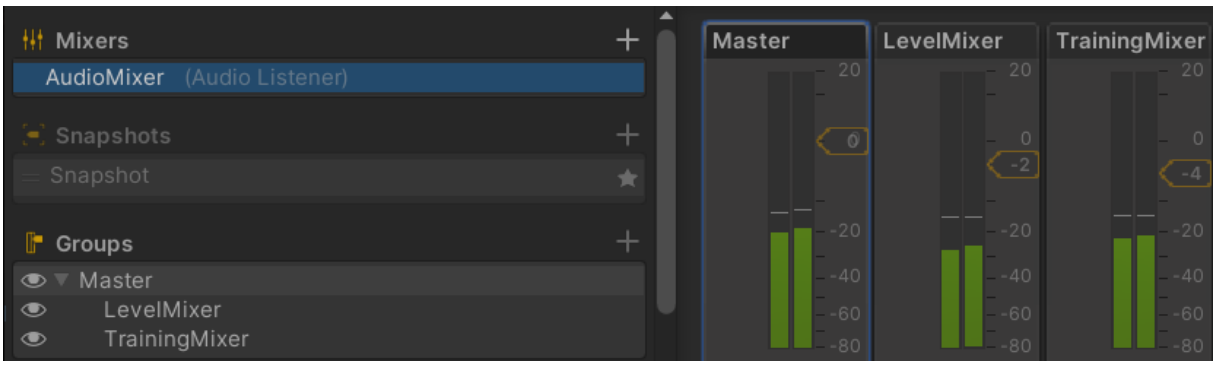


Slika 5.24. Prikaz glavnog izbornika



Slika 5.25. Prikaz opcija

Slikovne tipke sadrže za komponentu skriptu *LevelSelector*. U toj skripti je definirana funkcija *LoadLevel* koja prima varijablu tipa *string.* Ta varijabla predstavlja naziv scene razine. Odabirom slikovnih tipki od jedan do tri poziva se funkcija *LoadLevel* s pripadajućim nazivom scene i pokreće se razina. Objekt klizača sadrži komponentu, skriptu *AudioManager* koja upravlja glasnoćom audio isječaka. Kako bi se upravljalo glasnoćom potrebno je u Unity prozoru *Audio Mixer* izložiti varijablu glasnoće (engl. *volume*). Također u tom prozoru namještene su razine grupa audio isječaka (Slika 5.26). Glavni audio mikser predaje se objektu *BackgroundAudio* u sceni glavnog izbornika. Sukladno tome *LevelMixer* predaje se objektu *BackgroundAudio* u sceni razina, a *TrainingMixer* u sceni treninga.



Slika 5.26. Prikaz prozora Audio Mixer u Unitiyu

Klasi *AudioManager* predaje se objekt klizača i objekt audio miksera. Pri pokretanju scene provjerava se postoji li pohranjena vrijednost glasnoće pozivom funkcije *HasKey* na objektu klase *PlayerPrefs*. Klasa *PlayerPrefs* skladišti vrijednost glasnoće između igranja igre, tako da igra bude jednako glasna kao što je bila pri prijašnjem igranju. Za pohranu i učitavanje te vrijednosti korištene su funkcije *Load* i *Save*. Funkcija *SetVolume* služi za postavljanje granice glasnoće na objektu *AudioMixer*. Koristi se formula za logaritam od vrijednosti glasnoće (engl. *volume*) po bazi 10, i to se sve množi s 20 kako bi prijelazi u glasnoći bili blaži (Programski kod 5.20.). Pri učitavanju scene glazba počinje od početka. Za scenu glavnog izbornika korišten je audio isječak naziva „This Minimal Technology (Pure)“ izdan od korisnika „Coma-Media“, za scenu razina „Anime Beginings“ izdan od strane korisnika „PHANTASTICBEATS“, a za scenu treniranja „viyn – cotton candy (loop version)“ od korisnika „viyn“ (Prilog 5.2). Za prelazak iz opcija u glavni izbornik koristi se tipka. Ona je zadnji objekt koji se nalazi na prikazu opcija, odnosno tipka za povratak (engl. *back*). Pritiskom ove tipke objekt *Options* deaktivira se, a objekt *MainMenu* postavlja se kao aktivan.

Linija Kod

```
1:      using System;
2:      using UnityEngine;
3:      using UnityEngine.Audio;
4:      using UnityEngine.UI;
5:
6:      public class AudioManager : MonoBehaviour
7:      {
8:          [SerializeField] private Slider volumeSlider;
9:          [SerializeField] private AudioMixer audioMixer;
10:         void Start()
11:         {
12:             if (!PlayerPrefs.HasKey("volume"))
13:             {
14:                 PlayerPrefs.SetFloat("volume", 1);
15:             }
16:             Load();
17:         }
18:         public void SetVolume(float volume)
19:         {
20:             audioMixer.SetFloat("volume", Mathf.Log10(volume) * 20);
21:             Save();
22:         }
23:         private void Load()
24:         {
25:             volumeSlider.value = PlayerPrefs.GetFloat("volume");
26:         }
27:         private void Save()
28:         {
29:             PlayerPrefs.SetFloat("volume", volumeSlider.value);
30:         }
31:         public static implicit operator AudioManager(DontDestroyAudio v)
32:         {
33:             throw new NotImplementedException();
34:         }
35:     }
```

Programski kod 5.20. Klasa AudioManager

6. ZAKLJUČAK

Ovim radom istražena je primjena ML-Agents alata u Unity višeplatformskom softveru za treniranje agenata koristeći tehnike strojnog učenja. U Unityju izrađene su dvije okoline za treniranje agenta. Jednostavna okolina je korištena za izbor uspješnijeg algoritma, a napredna okolina se koristila za daljnje treniranje agenta. Uspoređeni su PPO i SAC algoritmi koji su korišteni za podržano strojno učenje. Iako SAC algoritam ima mogućnost učenja iz prijašnjih iskustava koja su spremljena u međuspremnik učinkovitiji se ispostavio PPO algoritam. Glavna karakteristika koja je pridonijela učinkovitosti PPO algoritma je njegovo trajanje. On ima jednostavnu implementaciju i teži što manjim odstupanjima trenutnog ponašanja od prethodnog pri ažuriranju. Rezultat treniranja agenta u okolinama za treniranje je istrenirani agent za prelazak razina platformske igre.

Nakon treniranja agenta, izrađena je platformska igra u cijelosti. Ona se sastoji od glavnog izbornika kojim se mogu izmijeniti opcije igre, pokrenuti razine na kojima se igrač natječe s agentom ili razina treniranja. Korisnik u razini treniranja može poboljšavati svoje sposobnosti bez pritiska agenta. Iz glavnog izbornika se može prijeći u opcije gdje se izravno može pokrenuti željena razina ili promijeniti glasnoća audio isječaka. Pozadina glavnog izbornika se kreće, što odmah nagovještava ugođaj igre. Na razinama koje korisnik može igrati, korišten je efekt paralaksa te su za objekte agenta, igrača i zamki napravljene animacije kako bi ugođaj kretanja bio ljepši.

Kao što je spomenuto u poglavlju treniranja agenta u naprednoj okolini, mogao bi se poboljšati položaj platformi odjeljka koje se nalaze preblizu krajnjeg odjeljka razine. S druge strane, ako bi se ostavili postojeći odjeljci trebalo bi se nastaviti s treniranjem agenta. Bilo bi potrebno promijeniti nagrade za akcije koje agent izvršava kako bi se ako zapne na vanjskom dijelu odjeljka cilja mogao vratiti nazad i uspješno prijeći razinu. Igra bi se također mogla proširiti dodavanjem novih razina ili interaktivnih objekata.

LITERATURA

[1] ,,Umjetna Inteligencija,“ [Mrežno]. Dostupno na: https://umjetna-inteligencija.webnode.page/. [Pokušaj pristupa 18. 6. 2023.].

[2] Europski parlament, ,,Što je umjetna inteligencija i kako se upotrebljava,“ 4. 9. 2020. [Mrežno]. Dostupno na: https://www.europarl.europa.eu/news/hr/headlines/society/20200827STO85804/sto-je-umjetna-inteligencija-i-kako-se-upotrebljava. [Pokušaj pristupa 18. 6. 2023.].

[3] D. Brozović, A. Kovačec, S. Ravlić, ,,Hrvatska enciklopedija, mrežno izdanje,“ 2021. [Mrežno]. Dostupno na: https://enciklopedija.hr/natuknica.aspx?ID=63150. [Pokušaj pristupa 18. 6. 2023.].

[4] N. Bolf, Kemija u industriji, svez. 70, Hrvatsko društvo kemijskih inženjera i tehnologa, 2021., pp. 591-592.

[5] O. Jacobi, ,,The Future of machine learning in video games,“ aporia, 18. 6. 2023. [Mrežno]. Dostupno na: https://www.aporia.com/blog/the-future-of-machine-learning-in-video-games/. [Pokušaj pristupa 26. 6. 2023.].

[6] T. Hovsepyan, ,,Machine Learning in Gaming,“ plat.ai, 18. 8. 2022. [Mrežno]. Dostupno na: https://plat.ai/blog/machine-learning-in-gaming/. [Pokušaj pristupa 27. 6. 2023.].

[7] Unity, ,,Unity Machine Learning Agents,“ [Mrežno]. Dostupno na: https://unity.com/products/machine-learning-agents. [Pokušaj pristupa 27. 6. 2023.].

[8] E. Teng, E. Vckay, J. Harper, Y. Gao, ,,Unity ML-Agents Toolkit v0.8: Faster training on real games,“ 15. 4. 2019. [Mrežno]. Dostupno na: https://blog.unity.com/engine-platform/unity-ml-agents-toolkit-v0-8-faster-training-on-real-games. [Pokušaj pristupa 27. 6. 2023.].

[9] Wikipedia, ,,Unity (game engine),“ [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Unity_(game_engine). [Pokušaj pristupa 25. 6. 2023.].

[10] M. Dealessandri, ,,What is the best game engine: is Unity right for you?,“ GamesIndustry.biz, 16. 1. 2020. [Mrežno]. Dostupno na: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you. [Pokušaj pristupa 25. 6. 2023.].

[11] D. Ankit, ,,Why Choose Unity 3D for Your Next Game Development Project?,“ MindInventory, 6. 4. 2022. [Mrežno]. Dostupno na: https://www.mindinventory.com/blog/unity-3d-game-development/. [Pokušaj pristupa 25. 6. 2023.].

[12] E. Cohen-Peckham, ,,How Unity built the world's most popular game engine,“ TechCrunch, 17. 10. 2019. [Mrežno]. Dostupno na: https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/. [Pokušaj pristupa 25. 6. 2023.].

[13] L. Schardon, ,,What is Unity? - A Guide for One of the Top Game Engines,“ ZENVA, [Mrežno]. Dostupno na: https://gamedevacademy.org/what-is-unity/. [Pokušaj pristupa 25. 6. 2023.].

[14] M. Mattar, J. Shih, V. Berges, C. Elion, C. Goy, ,,Announcing ML-Agents Unity Package v1.0,“ Unity, 12. 5. 2020. [Mrežno]. Dostupno na: https://blog.unity.com/engine-platform/announcing-ml-agents-unity-package-v1-0. [Pokušaj pristupa 28. 6. 2023.].

[15] Unity Tehnologies, „ML-Agents Toolkit Overview,“ [Mrežno]. Dostupno na: https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/. [Pokušaj pristupa 28. 6. 2023.].

[16] OpenAI, „Soft Actor-Critic,“ [Mrežno]. Dostupno na: https://spinningup.openai.com/en/latest/algorithms/sac.html. [Pokušaj pristupa 28. 6. 2023.].

[17] sidsen99, „A Brief Introduction to Proximal Policy Optimization,“ GeeksforGeeks, [Mrežno]. Dostupno na: https://www.geeksforgeeks.org/a-brief-introduction-to-proximal-policy-optimization/. [Pokušaj pristupa 28. 6. 2023.].

[18] J. Kanjilal, „Visual Studio Code: A fast, lightweight, cross-platform code editor,“ InfoWorld, 6. 5. 2015. [Mrežno]. Dostupno na: https://www.infoworld.com/article/2919555/visual-studio-code-a-fast-lightweight-and-cross-platform-code-editor.html. [Pokušaj pristupa 25. 6. 2023.].

[19] Visual Studio Code, „Getting Started,“ [Mrežno]. Dostupno na: https://code.visualstudio.com/docs. [Pokušaj pristupa 25. 6. 2023.].

[20] R. Sheldon, „C# (C-Sharp),“ TechTarget, [Mrežno]. Dostupno na: https://www.techtarget.com/whatis/definition/C-Sharp. [Pokušaj pristupa 25. 6. 2023.].

[21] M. Rouse, „C# (C Sharp),“ techopedia, 30. 4. 2020. [Mrežno]. Dostupno na: https://www.techopedia.com/definition/26272/c-sharp. [Pokušaj pristupa 25. 6. 2023.].

[22] GIMP, „A Brief (and Ancient) History of GIMP,“ [Mrežno]. Dostupno na: https://www.gimp.org/about/ancient_history.html. [Pokušaj pristupa 25. 6. 2023.].

[23] GIMP, „The Free & Open Source Image Editor,“ [Mrežno]. Dostupno na: https://www.gimp.org/. [Pokušaj pristupa 25. 6. 2023.].

[24] M. Muchmore, „GNU Image Manipulation Program (GIMP) Review,“ 22. 2. 2023. [Mrežno]. Dostupno na: https://www.pcmag.com/reviews/gnu-image-manipulation-program-gimp. [Pokušaj pristupa 25. 6. 2023.].

[25] S. Yegulalp, „What is TensorFlow? The machine learning library explained,“ InfoWorld, 3. 6. 2022. [Mrežno]. Dostupno na: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html. [Pokušaj pristupa 26. 6. 2023.].

[26] Skupina autora, „TensorFlow: A System for Large-Scale Machine Learning,“ [Mrežno]. Dostupno na: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf. [Pokušaj pristupa 26. 5. 2023.].

[27] Anaconda, „Our History,“ [Mrežno]. Dostupno na: https://www.anaconda.com/about-us. [Pokušaj pristupa 26. 6. 2023.].

[28] J. Helmus, „Understanding Conda and Pip,“ 28. 11. 2018. [Mrežno]. Dostupno na: https://www.anaconda.com/blog/understanding-conda-and-pip. [Pokušaj pristupa 26. 6. 2023.].

[29] Conda, „Getting Started with conda,“ [Mrežno]. Dostupno na: https://conda.io/projects/conda/en/latest/user-guide/getting-started.html. [Pokušaj pristupa 26. 6. 2023.].

[30] S. Brown, „Machine learning, explained,“ MIT Management Sloan School, 21. 4. 2021. [Mrežno]. Dostupno na: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. [Pokušaj pristupa 29. 6. 2023.].

[31] E. Burns, „machine learning,“ TechTarget, ožu. 2021. [Mrežno]. Dostupno na: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML. [Pokušaj pristupa 30. 6. 2023.].

[32] Unity Technologies, „Training Configuration File,“ [Mrežno]. Available: https://unity-technologies.github.io/ml-agents/Training-Configuration-File/#ppo-specific-configurations. [Pokušaj pristupa 16. 8. 2023.].

[33] GeeksforGeeks, „Shuffle a given array using Fisher–Yates shuffle Algorithm,“ 19. 12. 2022. [Mrežno]. Dostupno na: https://www.geeksforgeeks.org/shuffle-a-given-array-using-fisher-yates-shuffle-algorithm/. [Pokušaj pristupa 21. 8. 2023.].

SAŽETAK

Ovim radom opisane su korištene tehnologije i objašnjeni su elementi strojnog učenja koji su korišteni za treniranje agenta. Cilj je istrenirati agenta za rješavanje razina platformske igre koristeći ML-Agents alat. Izrađene su dvije okoline u Unityju za treniranje agenta, jedna jednostavna i jedna napredna. Jednostavna okolina je korištena za usporedbu PPO i SAC algoritama koji se koriste za podržano strojno učenje. Na temelju te usporedbe algoritama, efikasniji, PPO algoritam korišten je za treniranje agenta u naprednoj okolini. Uspješnost treniranja ispitana je na tri razine platformske igre. Rezultat ovog rada je uspješno istrenirani agent koji na temelju naučenog dolazi do cilja razine izbjegavajući zamke. Platformska igra se sastoji od tri razine na kojima se igrač može suprotstaviti agentu u natjecanju za pobjedu i jedne razine koja je korištena kao napredna okolina za treniranje, a sada na njoj igrač sam može trenirati.

Ključne riječi: Platformska igra, ML-Agents alat, Unity, Strojno učenje

ABSTRACT

Finding a platform game solution using machine learning

This work describes the technologies used and explains the machine learning elements used for agent training. The goal is to train an agent to solve a platform game level using the ML-Agents Toolkit. Two environments were created in Unity for training an agent, one simple and one advanced. A simple environment is used to compare PPO and SAC algorithms used for reinforcement learning. Based on this comparison of algorithms, the more efficient, PPO algorithm was used to train the agent in the advanced environment. The success of training was tested on three levels of platform games. The result of this work is a successfully trained agent that, based on what it has learned, reaches the level target while avoiding traps. The platform game consists of three levels where the player can face the agent in a competition to win and one level which is used as an advanced training environment and now the player can alone train on it.

Keywords: Platform game, ML-Agents Toolkit, Unity, Machine learning

ŽIVOTOPIS

Ana-Marija Katić rođena 18. siječnja 2000. godine u Osijeku. Od 2006. do 2008. pohađa Osnovnu školu Tin Ujević u Osijeku, a nakon drugog razreda mijenja mjesto stanovanja te nastavlja i završava osnovnoškolsko obrazovanje 2014. godine u Osnovnoj školi Frana Krste Frankopana. Zatim pohađa Prirodoslovno-matematičku gimnaziju u Osijeku gdje se prvi put susreće s programiranjem koristeći programski jezik C što postaje pokretač za buduće kontinuirano učenje programiranja. Pri završetku srednje škole 2018. godine redovno upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku koji završava 2021. godine. Nakon završetka preddiplomskog studija na istom fakultetu upisuje diplomski studij računarstva, smjer Programsko inženjerstvo.

Ana-Marija Katić

PRILOZI

P.5.1. Projekt platformske igre Gravity Agents s kodom i *.exe build*

Dostupan na: https://gitlab.com/Ana-Marija1/Gravity_Agents/

P.5.2. Audio isječci s mrežne stanice Pixabay

Audio isječak „This Minimal Technology (Pure)“ izdan od korisnika „Coma-Media“ dostupan je na: https://pixabay.com/music/corporate-this-minimal-technology-pure-12327/

Audio isječak „Anime Beginings“ izdan od korisnika „PHANTASTICBEATS“ dostupan je na: https://pixabay.com/music/cartoons-anime-beginings-139797/

Audio isječak „viyn – cotton candy (loop version)“ od korisnika „viyn“ dostupan je na: https://pixabay.com/music/future-bass-viyn-cotton-candy-loop-version-13253/

P.5.3. Slike objekata korištenih u igri s mrežne stranice Craftpix

Slike objekata likova agenta i igrača dostupne na: https://craftpix.net/freebies/free-3-cyberpunk-characters-pixel-art/

Slike objekata okoline dostupne na: https://craftpix.net/freebies/power-station-free-tileset-pixel-art/

Slike pozadina dostupne na: https://craftpix.net/freebies/free-city-backgrounds-pixel-art/